МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«ОРЛОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИМЕНИ И.С. ТУРГЕНЕВА»

Кафедра программной инженерии

**ОТЧЕТ**

по лабораторной работе № 2

на тему: «Структурное тестирование программного обеспечения»

по дисциплине: «Качество и тестирование программного обеспечения»

Вариант 18

Выполнил: Шорин В.Д. Шифр: 171406

Институт приборостроения, автоматизации и информационных технологий

Направление: 09.03.04 «Программная инженерия»

Группа: 71ПГ

Проверили: Ужаринский А.Ю., Конюхова О.В.

Отметка о зачете:

Дата: «\_\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2020 г.

Орел, 2020 г.

**Задание на лабораторную работу:**

У преподавателя есть список заданий, каждое из который имеет тип (теория/практика/блиц/прочие), тематику и уровень сложности. Постройте алгоритм для генерации списка билетов, такой, чтобы одновременно выполнялись условия: 1) в зависимости от внешних условий состав билетов меняется; 2) билет содержит одинаковое количество вопросов из разных тем, и одинаковое соотношение заданий разных типов (например, 2 теории и одна практика); 3) уровень сложности всех билетов приблизительно одинаковый.

**Основная программа**

Определение базового множества

1. 1 — 2 — 18 — 20
2. 1 – 2 – 3 – 4 – 5 – 6 – 7 – 8 – 9 – 10 – 12 – 6 – 13 – 14 — 15 — 18 — 19 — 18 — 20
3. -//- 14 — 16 — 18 — 19 — 18 — 20
4. -//- 14 — 16 — 17 — 2 — 18 — 19 — 18 — 20
5. 1 – 2 – 18 – 19 – 18 – 20 ( Не существует ИД)
6. 1 – 2 – 3 – 4 – 5 – 6 – 13 – 14 – 15 – 18 – 19 – 18 – 20 (Не существует ИД)
7. -//- 14 – 16 – 18 – 19 – 18 – 20 (Не существует ИД)
8. -//- 14 – 16 – 17 – 2 – 18 – 19 – 18 – 20 (Не существует ИД)

Тестовые варианты:

1. ИД: Массив задач пуст (tasks == null);

ОР: пустой вывод

1. ИД: Массив задач содержит равный список задач (одинаковое количество практик и теорий, разделенных по темам и уровням сложности) tasks > 0, после выполнения очередного цикла в массиве tasks осталось 2 задачи.

ОР: вывод билетов

1. ИД: Массив задач содержит равный список задач (одинаковое количество практик и теорий, разделенных по темам и уровням сложности) tasks > 0, после выполнения очередного цикла в массиве tasks осталось 0 задач.

ОР: вывод билетов

1. ИД: Массив задач содержит равный список задач (одинаковое количество практик и теорий, разделенных по темам и уровням сложности) tasks > 0, после выполнения очередного цикла в массиве tasks осталось больше 2 задач.

ОР: вывод билетов

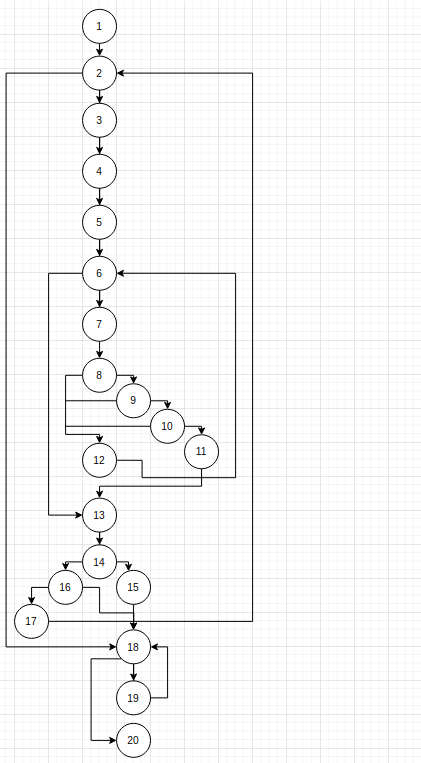


Рисунок 1 — Потоковый граф основной программы

**Функция GetTasksFromXML**

Определение базового множества

1) 1 — 2 — 14

2) 1 — 2 — 3 — 13 — 2 — 14

3) -//- 5 — 7 — 9 — 10 — 12 -//-

4) -//- 6 — 7 — 8 — 10 — 12 -//-

5) -//- 6 — 7 — 9 — 10 — 11 -//-

Тестовые варианты:

1. ИД: xRoot == null

ОР: возврат пустого списка

1. ИД: xRoot != null (Переменная содержит список пустых записей)

ОР: возврат списка задач

1. ИД: xRoot != null (Переменная содержит список записей о задачах), название очередной считанной дочерней записи «taskType»

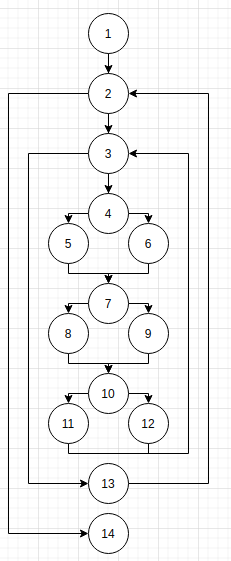
ОР: возврат списка задач

1. ИД: xRoot != null (Переменная содержит список записей о задачах), название очередной считанной дочерней записи «theme»

ОР: возврат списка задач

1. ИД: xRoot != null (Переменная содержит список записей о задачах), название очередной считанной дочерней записи «difficulty»

ОР: возврат списка задач

Рисунок 2 — Потоковый граф функции GetTasksFromXML

Код

«Program.cs»

using System;using System.Collections;using System.Collections.Generic;using System.Linq;using System.Xml;

using System.Xml.Linq;using Lab2.Classes;

namespace Lab2{

class Program {

static void Main(string[] args)

{

string path = "../../../tasks.xml";

// 1

List<Task> tasks = GetTasksFromXML(path);

List<Ticket> tickets = new List<Ticket>();

int ticketNumber = 1;

var random = new Random();

// 2

while (true) {

// 3

List<Task> ticketTasks = new List<Task>(); int randomTask = random.Next(tasks.Count);

// 4

// Add first task (any, any, any)

ticketTasks.Add(tasks[randomTask]); tasks.RemoveAt(randomTask);

// 5

randomTask = random.Next(tasks.Count); List<Task> copyTasks = tasks.ToList();

// 6

// Add 2rd task (!such in 1st task, !such in 1st task, sum == 4)

while (true) {

// 7

copyTasks[randomTask].PrintTask();

// 8 9 10

if (copyTasks[randomTask].TaskType != ticketTasks[0].TaskType

&& copyTasks[randomTask].Theme != ticketTasks[0].Theme

&& copyTasks[randomTask].Difficulty + ticketTasks[0].Difficulty == 4

) {

// 11

ticketTasks.Add(copyTasks[randomTask]); tasks.Remove(copyTasks[randomTask]);

break;

}

// 12

copyTasks.RemoveAt(randomTask);

randomTask = (copyTasks.Count == 0) ? 0 : random.Next(copyTasks.Count);

}

// 13

tickets.Add(new Ticket(ticketNumber, ticketTasks));

// 14

if (tasks.Count == 2) {

// 15

ticketNumber++;

ticketTasks = tasks.ToList();

tickets.Add(new Ticket(ticketNumber, ticketTasks));

break;

}

// 16

if (tasks.Count == 0) { break; }

// 17

ticketNumber++;

}

Console.WriteLine("Tickets");

// 18

foreach (var ticket in tickets) { ticket.PrintTicket(); }

//Console.ReadKey();

}

public static void CreateTasks(string pathToSave) {

string[] themes = {"theme1", "theme2", "theme3", "theme4"};

int[] difficulties = {1, 2, 3};

string[] taskTypes = {"theory", "practice"};

XDocument xDocument = new XDocument();

XElement tasks = new XElement("tasks");

foreach (var theme in themes) {

foreach (var difficulty in difficulties) {

foreach (var taskType in taskTypes) {

XElement xeTask = new XElement("task");

XElement xeTaskType = new XElement("taskType", taskType);

XElement xeTheme = new XElement("theme", theme);

XElement xeDifficulty = new XElement("difficulty", difficulty);

xeTask.Add(xeTaskType);

xeTask.Add(xeTheme);

xeTask.Add(xeDifficulty);

tasks.Add(xeTask);

}

}

}

xDocument.Add(tasks);

xDocument.Save(pathToSave);

}

public static List<Task> GetTasksFromXML(string pathToLoad) {

// 1

List<Task> tasks = new List<Task>();

XmlDocument xmlDocument = new XmlDocument();

xmlDocument.Load(pathToLoad);

XmlElement xRoot = xmlDocument.DocumentElement;

string taskType = "";

string theme = "";

int difficulty = 0;

// 2

foreach (XmlNode xmlNode in xRoot) {

// 3

foreach (XmlNode childNode in xmlNode.ChildNodes) {

// 4 // 5 // 6

taskType = (childNode.Name == "taskType") ? childNode.InnerText : taskType;

// 7 // 8 // 9

theme = (childNode.Name == "theme") ? childNode.InnerText : theme;

// 10 // 11 // 12

difficulty = (childNode.Name == "difficulty") ? Convert.ToInt32(childNode.InnerText) : difficulty;

}

// 13

tasks.Add(new Task(taskType, theme, difficulty));

}

// 14

return tasks;

}

}

}

«Task.cs»

using System;

namespace Lab2.Classes{

public class Task {

public string TaskType { get; private set;}

public string Theme { get; private set; }

public int Difficulty { get; private set; }

public Task(string type, string theme, int difficulty) {

TaskType = type;

Theme = theme;

Difficulty = difficulty;

}

public Task() { throw new NotImplementedException(); }

public void PrintTask() {

Console.WriteLine($"TaskType: {TaskType} " + $"Theme: {Theme} " +$"Difficulty: {Difficulty} ");

}

}

}

«Ticket.cs»

using System;

using System.Collections.Generic;

using Lab2.Classes;

namespace Lab2.Classes{

public class Ticket {

public int Number { get; private set; }

public List<Task> Tasks { get; private set; }

public Ticket(int number, List<Task> tasks) { Tasks = tasks; Number = number; }

public void PrintTicket() {

Console.WriteLine($"Number: {Number}");

foreach (var task in Tasks) { task.PrintTask(); }

Console.WriteLine();

}

}

}