Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГТУ»)**

**Институт компьютерных систем и информационной безопасности**

**Кафедра информационных систем и программирования**

**ЛАБОРАТОРНАЯ РАБОТА №8**

**Дисциплина: тестирование и отладка программного обеспечения**

Работу выполнил: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Я.С Давков

Направление подготовки: 09.03.04 Программная инженерия

Преподаватель: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.Г. Волик

**1 Цель работы**

Изучить подход к автоматизации процесса тестирования с использованием тестовых двойников (Test Doubles).

**2 Краткая теория**

Модульное тестирование представляет собой это тестирование программы на уровне отдельно взятых модулей, функций или классов. Однако между модулями и классами существуют взаимные зависимости. Когда один модуль или класс зависти от другого возникает проблема в тестировании зависимого класса. При возникновении ошибки в таких случаях не всегда можно сказать однозначно, кто является виновником проблемы: недостаточно протестированный класс нижнего уровня или сам тестируемый класс. Для исключения подобных эффектов необходимо иметь возможность однозначного определения источника проблемы. В этом случае для успешного осуществления модульного тестирования необходимо эффективное замещение серверов компонентов в тестовых целях. Объекты-заместители обычно называются заглушками (stubs), или фиктивными объектами (mocks). Термин «тестовый двойник», применяется как общее имя для объектов, используемых для замены настоящих компонентов в целях тестирования. Тестовый двойник является общим термином, а для конкретных реализаций используются специальные имена. Различают пять типов тестовых двойников (таблица 8.1). Хотя в теории эти типы четко разделяются, разница размывается на практике.

**3 Задание**

1) Создать класс, реализующий граф/дерево, хранящий информацию о структуре графа на основе некоторого внутреннего представления указанного в задании из пункта 5.1. 2) В созданном классе реализовать один из методов обработки графа/дерева в соответствии с вариантом задания из пункта 5.2. 3) В качестве источника данных использовать класс, реализующий чтение из файла произвольного формата и возвращающего данные в виде, указанном в варианте задания из пункта 5.1. 132 4) Реализовать возможность сохранения данных в файл произвольного формата, с помощью отдельного класса, в формате, определяемом в соответствии с вариантом задания в пункте 5.1. 5) Для обработки файла создать отдельный класс/классы, реализующий методы загрузки из файла (возвращающий считанные данных в определенном формате) и сохранения в файл (сохраняющий матрицу в определенном формате, который может отличаться от внутреннего представления графа). 6) Протестировать класс для работы с заданным форматом файлов (при тестировании класса использовать подмену класса на StringReader /StringWriter для доступа к строковому потоку с последующей проверкой поученной выходной строки). 7) Протестировать класс, реализующий непосредственные вычисления (при получении данных и сохранения использовать заглушки и фиктивные объекты). 8) Оформить отчёт.

**4 Выполнение работы**

Мой вариант – 4, реализуемый алгоритм - Проверка на наличие циклов в графе, Входной формат - Матрица смежности, Внутреннее представление - Матрица смежности, Выходной формат - Матрица инцидентности

Листинг кода:

public class MyGraph  
{  
 public int[,] AdjacencyMatrix { get; }  
  
 public int VertexCount => AdjacencyMatrix.GetLength(0);  
  
 public MyGraph(int[,] adjacencyMatrix)  
 {  
 AdjacencyMatrix = adjacencyMatrix ?? throw new ArgumentNullException(nameof(adjacencyMatrix));  
 }  
  
 public bool HasCycle()  
 {  
 bool[] visited = new bool[VertexCount];  
 bool[] stack = new bool[VertexCount];  
  
 for (int i = 0; i < VertexCount; i++)  
 if (DetectCycleDFS(i, visited, stack))  
 return true;  
  
 return false;  
 }  
  
 private bool DetectCycleDFS(int v, bool[] visited, bool[] stack)  
 {  
 if (!visited[v])  
 {  
 visited[v] = true;  
 stack[v] = true;  
  
 for (int j = 0; j < VertexCount; j++)  
 {  
 if (AdjacencyMatrix[v, j] == 1)  
 {  
 if (!visited[j] && DetectCycleDFS(j, visited, stack))  
 return true;  
 else if (stack[j])  
 return true;  
 }  
 }  
 }  
  
 stack[v] = false;  
 return false;  
 }  
  
 public int[,] ToIncidenceMatrix()  
 {  
 var edges = new List<(int from, int to)>();  
  
 for (int i = 0; i < VertexCount; i++)  
 for (int j = 0; j < VertexCount; j++)  
 if (AdjacencyMatrix[i, j] == 1)  
 edges.Add((i, j));  
  
 int[,] incidence = new int[VertexCount, edges.Count];  
  
 for (int k = 0; k < edges.Count; k++)  
 {  
 incidence[edges[k].from, k] = 1;  
 incidence[edges[k].to, k] = -1;  
 }  
  
 return incidence;  
 }  
}

public class MatrixGraphWriter : IGraphWriter  
{  
 public void Write(TextWriter writer, int[,] matrix)  
 {  
 int rows = matrix.GetLength(0);  
 int cols = matrix.GetLength(1);  
  
 for (int i = 0; i < rows; i++)  
 {  
 for (int j = 0; j < cols; j++)  
 {  
 writer.Write(matrix[i, j]);  
 if (j < cols - 1) writer.Write(' ');  
 }  
 writer.WriteLine();  
 }  
 }  
}

public class MatrixGraphReader : IGraphReader  
{  
 public int[,] Read(TextReader reader)  
 {  
 var lines = new List<string>();  
 string? line;  
 while ((line = reader.ReadLine()) != null)  
 {  
 if (!string.IsNullOrWhiteSpace(line))  
 lines.Add(line.Trim());  
 }  
  
 int n = lines.Count;  
 int[,] matrix = new int[n, n];  
  
 for (int i = 0; i < n; i++)  
 {  
 var parts = lines[i].Split(' ', StringSplitOptions.*RemoveEmptyEntries*);  
 for (int j = 0; j < n; j++)  
 matrix[i, j] = int.Parse(parts[j]);  
 }  
  
 return matrix;  
 }  
}

public interface IGraphWriter  
{  
 void Write(TextWriter writer, int[,] matrix);  
}

public interface IGraphReader  
{  
 int[,] Read(TextReader reader);  
}

public class GraphSaver  
{  
 private readonly IGraphWriter writer;  
  
 public GraphSaver(IGraphWriter writer)  
 {  
 this.writer = writer ?? throw new ArgumentNullException(nameof(writer));  
 }  
  
 public void Save(string path, int[,] matrix)  
 {  
 string dir = Path.GetDirectoryName(path);  
 if (!string.IsNullOrEmpty(dir) && !Directory.Exists(dir))  
 Directory.CreateDirectory(dir);  
  
 using var sw = new StreamWriter(path);  
 writer.Write(sw, matrix);  
 }  
}

public class GraphFileProcessor  
{  
 private readonly IGraphReader reader;  
 private readonly IGraphWriter writer;  
  
 public GraphFileProcessor(IGraphReader reader, IGraphWriter writer)  
 {  
 this.reader = reader ?? throw new ArgumentNullException(nameof(reader));  
 this.writer = writer ?? throw new ArgumentNullException(nameof(writer));  
 }  
  
 public int[,] Load(string path)  
 {  
 using var sr = new StreamReader(path);  
 return reader.Read(sr);  
 }  
  
 public void Save(string path, int[,] matrix)  
 {  
 using var sw = new StreamWriter(path);  
 writer.Write(sw, matrix);  
 }  
}

Тесты выполнены с использованием заглушек, запустим и проверим что все работает.

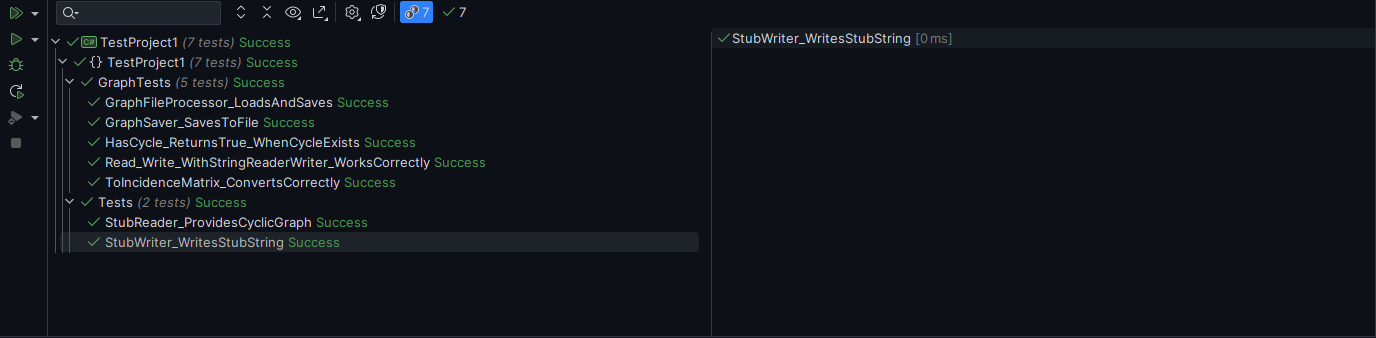


Рисунок 1 – запуск тестом.