МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ им. Р.Е.АЛЕКСЕЕВА

Институт радиоэлектроники и информационных технологий
Кафедра «Вычислительные системы и технологии»

Лабораторная работа №1
по дисциплине
Тестирование программного обеспечения

РУКОВОДИТЕЛЬ:

_____ Скорынин С.С.

СТУДЕНТ:

_____ Петрова П.Е..

21-ПО

Нижний Новгород 2025

Цель: Изучить основы разработки модульных тестов. Получить навыки работы со средствами тестирования.

Задание:

Изучить средства тестирования, доступные в Visual Studio – Unit Testing Framework, NUnit. Разработать набор unit-тестов для алгоритма в соответствии с номером варианта. Реализовать алгоритм в соответствии с номером варианта на любом языке, поддерживаемом платформой .NET. Обеспечить максимально возможное покрытие кода тестами.

Вариант 17:

16	Выборка (подразумевает предварительное построение	бинарным поиском после сортировки (можно использовать библиотечную функцию)	IEnumerable<T> Select<T>(IEnumerable<T> sequence, Func<T, bool> condition)
17		из B-дерева	
18		из B+дерева	

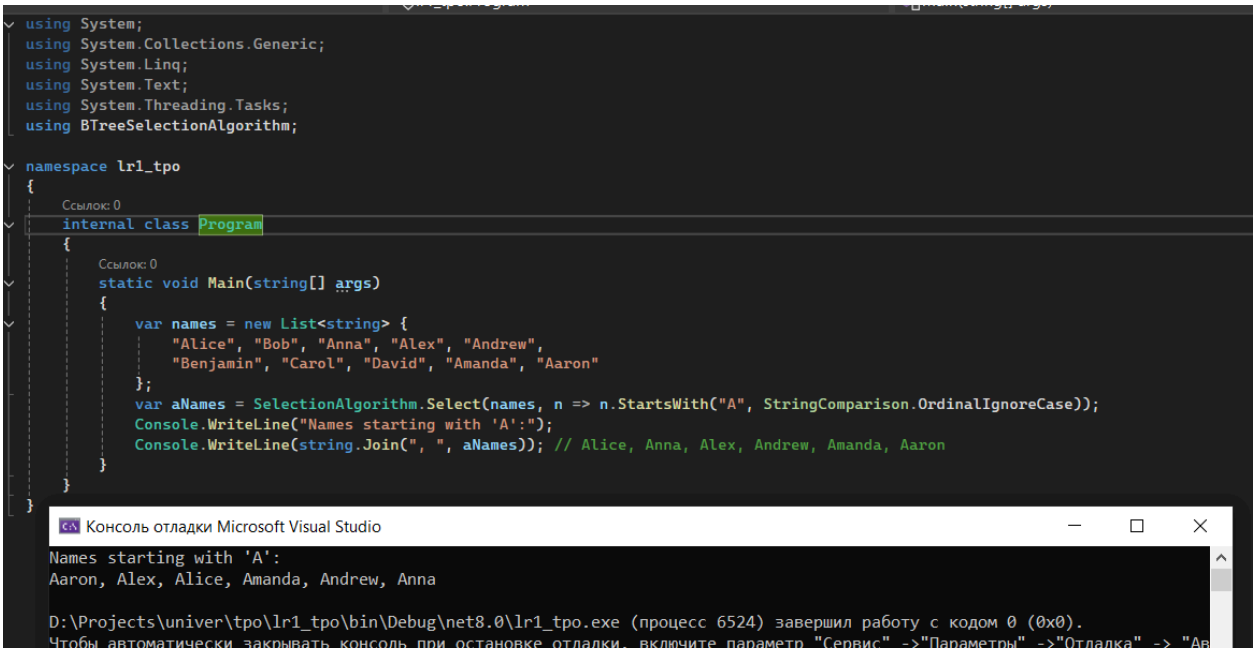
Работа выполнена на языке C# 12 в Visual Studio 2022 с использование NUnit, Moq, Coverlet.

```
<PropertyGroup>
  <TargetFramework>net8.0</TargetFramework>
  <ImplicitUsings>enable</ImplicitUsings>
  <Nullable>enable</Nullable>

  <IsPackable>false</IsPackable>
  <IsTestProject>true</IsTestProject>
</PropertyGroup>

<ItemGroup>
  <PackageReference Include="coverlet.collector" Version="6.0.4">
    <PrivateAssets>all</PrivateAssets>
    <IncludeAssets>runtime; build; native; contentfiles; analyzers; buildtransitive</IncludeA
  </PackageReference>
  <PackageReference Include="Microsoft.NET.Test.Sdk" Version="17.8.0" />
  <PackageReference Include="Moq" Version="4.20.72" />
  <PackageReference Include="NUnit" Version="3.14.0" />
  <PackageReference Include="NUnit.Analyzers" Version="3.9.0" />
  <PackageReference Include="NUnit3TestAdapter" Version="4.5.0" />
</ItemGroup>
```

Результат работы алгоритма:



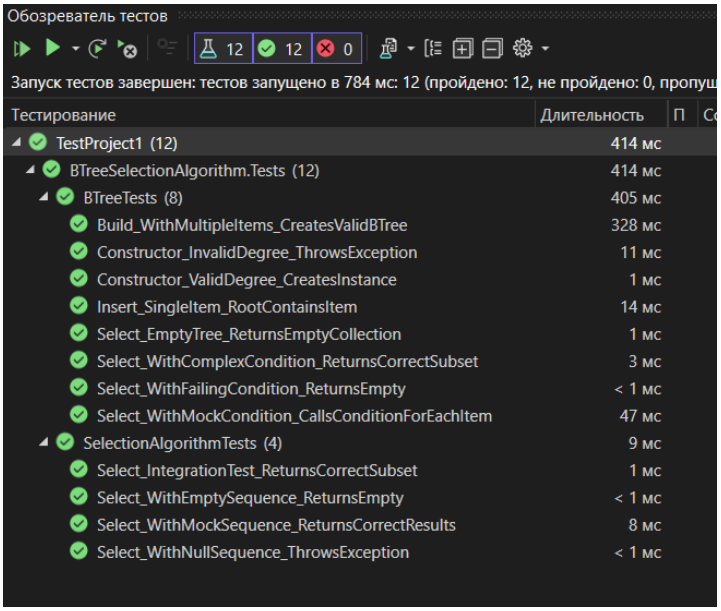
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using BTreeSelectionAlgorithm;

namespace lr1_tpo
{
    Ссылок: 0
    internal class Program
    {
        Ссылок: 0
        static void Main(string[] args)
        {
            var names = new List<string> {
                "Alice", "Bob", "Anna", "Alex", "Andrew",
                "Benjamin", "Carol", "David", "Amanda", "Aaron"
            };
            var aNames = SelectionAlgorithm.Select(names, n => n.StartsWith("A", StringComparison.OrdinalIgnoreCase));
            Console.WriteLine("Names starting with 'A':");
            Console.WriteLine(string.Join(", ", aNames)); // Alice, Anna, Alex, Andrew, Amanda, Aaron
        }
    }
}
```

```
Консоль отладки Microsoft Visual Studio
Names starting with 'A':
Aaron, Alex, Alice, Amanda, Andrew, Anna

D:\Projects\univer\tpo\lr1_tpo\bin\Debug\net8.0\lr1_tpo.exe (процесс 6524) завершил работу с кодом 0 (0x0).
Чтобы автоматически закрывать консоль при остановке отладки, включите параметр "Сервис" ->"Параметры" ->"Отладка" -> "Ав
```

Результат тестирования:



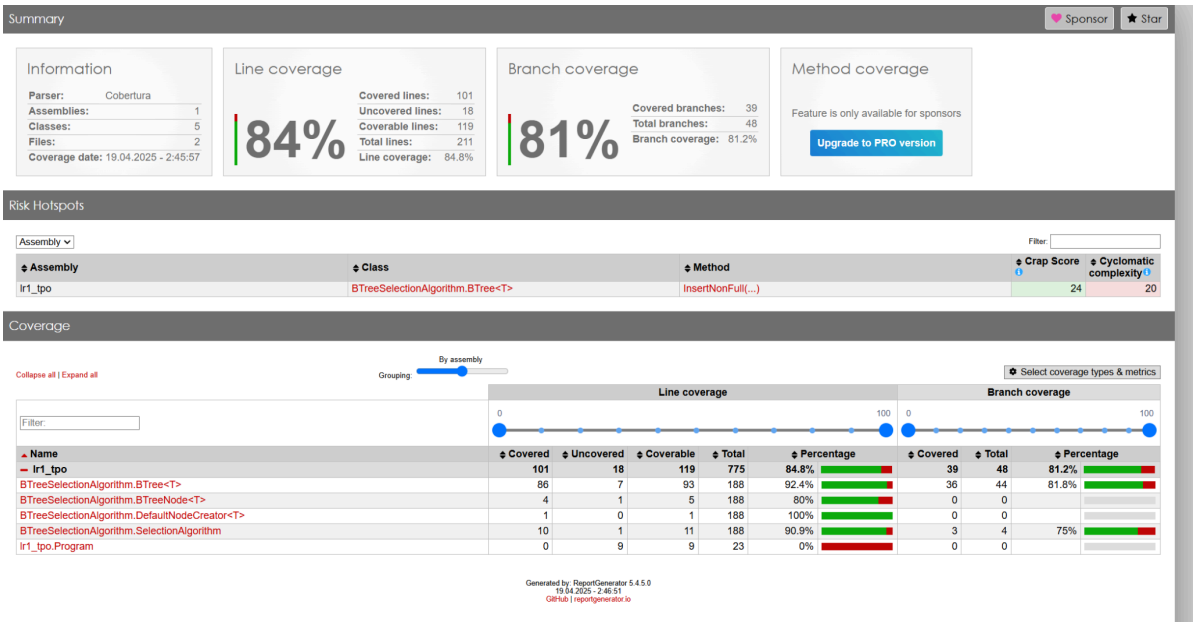
Обозреватель тестов

12 12 0

Запуск тестов завершен: тестов запущено в 784 мс: 12 (пройдено: 12, не пройдено: 0, пропущ

Тестирование	Длительность	П	С
TestProject1 (12)	414 мс		
BTreeSelectionAlgorithm.Tests (12)	414 мс		
BTreeTests (8)	405 мс		
Build_WithMultipleItems_CreatesValidBTree	328 мс		
Constructor_InvalidDegree_ThrowsException	11 мс		
Constructor_ValidDegree_CreatesInstance	1 мс		
Insert_SingleItem_RootContainsItem	14 мс		
Select_EmptyTree_ReturnsEmptyCollection	1 мс		
Select_WithComplexCondition_ReturnsCorrectSubset	3 мс		
Select_WithFailingCondition_ReturnsEmpty	< 1 мс		
Select_WithMockCondition_CallsConditionForEachItem	47 мс		
SelectionAlgorithmTests (4)	9 мс		
Select_IntegrationTest_ReturnsCorrectSubset	1 мс		
Select_WithEmptySequence_ReturnsEmpty	< 1 мс		
Select_WithMockSequence_ReturnsCorrectResults	8 мс		
Select_WithNullSequence_ThrowsException	< 1 мс		

Покрытие кода тестами:



Код алгоритма:

Btree.cs

```
using System;
using System.Collections.Generic;
using System.Linq;

namespace BTreeSelectionAlgorithm
{
    // Узел B-дерева
    public interface IBTreeNode<T> where T : IComparable<T>
    {
        List<T> Keys { get; }
        List<IBTreeNode<T>> Children { get; }
        bool IsLeaf { get; }
        void AddKey(T key);
        void AddChild(IBTreeNode<T> node);
    }

    public class BTreeNode<T> : IBTreeNode<T> where T : IComparable<T>
    {
```

```
    public List<T> Keys { get; } = new List<T>();
    public List<IBTreeNode<T>> Children { get; } = new List<IBTreeNode<T>>();
    public bool IsLeaf => Children.Count == 0;

    public void AddKey(T key) => Keys.Add(key);
    public void AddChild(IBTreeNode<T> node) => Children.Add(node);
}

public class DefaultNodeCreator<T> : BTree<T>.INodeCreator<T>
where T : IComparable<T>
{
    public IBTreeNode<T> CreateNode() => new BTreeNode<T>();
}

// B-дерево
public class BTree<T> where T : IComparable<T>
{
    private readonly int _degree;
    private IBTreeNode<T> _root;
    private readonly INodeCreator<T> _nodeCreator;

    public interface INodeCreator<U> where U : IComparable<U>
    {
        IBTreeNode<U> CreateNode();
    }

    public BTree(int degree, IBTreeNode<T> rootNode, INodeCreator<T> nodeCreator)
    {
        if (degree < 2)
            throw new ArgumentException("Degree must be at least 2", nameof(degree));
        _degree = degree;
        _root = rootNode;
        _nodeCreator = nodeCreator;
```

```
}

// Сборка
public void Build(IEnumerable<T> sequence)
{
    if (sequence == null)
        throw new ArgumentNullException(nameof(sequence));

    foreach (var item in sequence)
    {
        Insert(item);
    }
}

// Добавление ключа в дерево
private void Insert(T key)
{
    if (_root.Keys.Count == (2 * _degree) - 1)
    {
        var newRoot = _nodeCreator.CreateNode();
        newRoot.AddChild(_root);
        SplitChild(newRoot, 0, _nodeCreator);
        _root = newRoot;
    }
    InsertNonFull(_root, key);
}

// Поиск места для вставки
private void InsertNonFull(IBTreeNode<T> node, T key)
{
    int i = node.Keys.Count - 1;
    if (node.IsLeaf)
    {
```

```
            while (i >= 0 && key.CompareTo(node.Keys[i]) < 0)
            {
                i--;
            }
            node.Keys.Insert(i + 1, key);
        }
        else
        {
            while (i >= 0 && key.CompareTo(node.Keys[i]) < 0)
            {
                i--;
            }
            i++;
            if (node.Children[i].Keys.Count == (2 * _degree) - 1)
            {
                SplitChild(node, i, _nodeCreator);
                if (key.CompareTo(node.Keys[i]) > 0)
                {
                    i++;
                }
            }
            InsertNonFull(node.Children[i], key);
        }
    }

    // Разделение потомка на два узла
    private void SplitChild(IBTreeNode<T> parentNode,
                int childIndex,
                INodeCreator<T> creator)
    {
        var child = parentNode.Children[childIndex];
        var newNode = creator.CreateNode();
        parentNode.Keys.Insert(childIndex, child.Keys[_degree - 1]);
```

```
    parentNode.Children.Insert(childIndex + 1, newNode);

    newNode.Keys.AddRange(child.Keys.GetRange(_degree, _degree - 1));
    child.Keys.RemoveRange(_degree - 1, _degree);

    if (!child.IsLeaf)
    {
        newNode.Children.AddRange(child.Children.GetRange(_degree, _degree));
        child.Children.RemoveRange(_degree, _degree);
    }
}

// Поиск
public IEnumerable<T> Select(Func<T, bool> condition)
{
    return SelectInternal(_root, condition);
}

private IEnumerable<T> SelectInternal(IBTreeNode<T> node, Func<T, bool> condition)
{
    if (node == null) yield break;

    for (int i = 0; i < node.Keys.Count; i++)
    {
        if (!node.IsLeaf)
        {
            foreach (var item in SelectInternal(node.Children[i], condition))
            {
                yield return item;
            }
        }

        if (condition(node.Keys[i]))
```

```
            {
                yield return node.Keys[i];
            }
        }

        if (!node.IsLeaf)
        {
            foreach (var item in SelectInternal(node.Children.Last(), condition))
            {
                yield return item;
            }
        }
    }
}

public static class SelectionAlgorithm
{
    public static IEnumerable<T> Select<T>(IEnumerable<T> sequence, Func<T, bool> condition)
        where T : IComparable<T>
    {
        if (sequence == null)
            throw new ArgumentNullException(nameof(sequence));
        if (condition == null)
            throw new ArgumentNullException(nameof(condition));

        var creator = new DefaultNodeCreator<T>();
        var root = creator.CreateNode();

        const int degree = 100;

        var btree = new BTree<T>(degree, root, creator);

        btree.Build(sequence);
```

```
            return btree.Select(condition);
        }
    }
}
```

Код тестов:

UnitTest1.cs

```
using NUnit.Framework;
using Moq;
using System;
using System.Collections.Generic;
using System.Linq;

namespace BTreeSelectionAlgorithm.Tests
{
    [TestFixture]
    public class BTreeTests
    {
        private const int Degree = 2;
        private BTree<int> _btree;
        private Mock<BTree<int>.INodeCreator<int>> _mockNodeCreator;
        private Mock<IBTreeNode<int>> _mockRootNode;

        [SetUp]
        public void Setup()
        {
            _mockNodeCreator = new Mock<BTree<int>.INodeCreator<int>>();
            _mockRootNode = new Mock<IBTreeNode<int>>();

            _mockRootNode.SetupGet(x => x.Keys).Returns(new List<int>());
            _mockRootNode.SetupGet(x => x.Children).Returns(new List<IBTreeNode<int>>());
            _mockRootNode.SetupGet(x => x.IsLeaf).Returns(true);
```

```
    _mockNodeCreator.Setup(x => x.CreateNode()).Returns(() =>
    {
        var mockNode = new Mock<IBTreeNode<int>>();
        mockNode.SetupGet(n => n.Keys).Returns(new List<int>());
        mockNode.SetupGet(n => n.Children).Returns(new List<IBTreeNode<int>>());
        mockNode.SetupGet(n => n.IsLeaf).Returns(true);
        return mockNode.Object;
    });

    _btree = new BTree<int>(Degree, _mockRootNode.Object, _mockNodeCreator.Object);
}

[Test]
public void Constructor_ValidDegree_CreatesInstance()
{
    var creator = new DefaultNodeCreator<int>();
    var root = creator.CreateNode();
    Assert.DoesNotThrow(() => new BTree<int>(2, root, creator));
}

[Test]
public void Constructor_InvalidDegree_ThrowsException()
{
    var creator = new DefaultNodeCreator<int>();
    var root = creator.CreateNode();
    Assert.Throws<ArgumentException>(() => new BTree<int>(1, root, creator));
}

[Test]
public void Insert_SingleItem_RootContainsItem()
{
    _btree.Build(new[] { 42 });
```

```
    var result = _btree.Select(x => true).ToList();
    Assert.That(1, Is.EqualTo(result.Count));
    Assert.That(42, Is.EqualTo(result[0]));
}


[Test]
public void Select_EmptyTree_ReturnsEmptyCollection()
{
    var result = _btree.Select(x => true);
    CollectionAssert.IsEmpty(result);
}

[Test]
public void Select_WithMockCondition_CallsConditionForEachItem()
{
    var mockCondition = new Mock<Func<int, bool>>();
    mockCondition.Setup(x => x(It.IsAny<int>())).Returns(true);

    _btree.Build(new[] { 1, 2, 3 });
    _btree.Select(mockCondition.Object).ToList();

    mockCondition.Verify(x => x(1), Times.Once);
    mockCondition.Verify(x => x(2), Times.Once);
    mockCondition.Verify(x => x(3), Times.Once);
}

[Test]
public void Select_WithFailingCondition_ReturnsEmpty()
{
    _btree.Build(new[] { 1, 2, 3 });
    var result = _btree.Select(x => false).ToList();
    CollectionAssert.IsEmpty(result);
```

```
}

[Test]
public void Build_WithMultipleItems_CreatesValidBTree()
{
    // Arrange
    var creator = new DefaultNodeCreator<int>();
    var root = creator.CreateNode();
    var btree = new BTree<int>(Degree, root, creator);
    var sequence = new[] { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

    // Act
    btree.Build(sequence);
    var result = btree.Select(x => true).ToList();

    // Assert
    Assert.That(result, Has.Count.EqualTo(10));
    CollectionAssert.AreEquivalent(sequence, result);
}

[Test]
public void Select_WithComplexCondition_ReturnsCorrectSubset()
{
    // Arrange
    var creator = new DefaultNodeCreator<int>();
    var root = creator.CreateNode();
    var btree = new BTree<int>(Degree, root, creator);
    btree.Build(Enumerable.Range(1, 100));

    // Act
    var result = btree.Select(x => x % 3 == 0 && x % 5 == 0).ToList();

    // Assert
```

```
        Assert.That(result, Has.Count.EqualTo(6)); // 15, 30, 45, 60, 75, 90
        CollectionAssert.Contains(result, 15);
        CollectionAssert.Contains(result, 90);
        CollectionAssert.DoesNotContain(result, 10);
    }
}

[TestFixture]
public class SelectionAlgorithmTests
{
    [Test]
    public void Select_WithMockSequence_ReturnsCorrectResults()
    {
        var mockSequence = new Mock<IEnumerable<int>>();
        mockSequence.Setup(x => x.GetEnumerator())
                .Returns(() => ((IEnumerable<int>)new[] { 1, 2, 3, 4, 5 }).GetEnumerator());

        var result = SelectionAlgorithm.Select(mockSequence.Object, x => x > 3).ToList();

        CollectionAssert.AreEquivalent(new[] { 4, 5 }, result);
    }

    [Test]
    public void Select_WithEmptySequence_ReturnsEmpty()
    {
        var result = SelectionAlgorithm.Select(Enumerable.Empty<int>(), x => true);
        CollectionAssert.IsEmpty(result);
    }

    [Test]
    public void Select_WithNullSequence_ThrowsException()
    {
        Assert.Throws<ArgumentNullException>(() =>
```

```
            SelectionAlgorithm.Select<int>(null, x => true));
        }

        [Test]
        public void Select_IntegrationTest_ReturnsCorrectSubset()
        {
            var data = Enumerable.Range(1, 100);
            var result = SelectionAlgorithm.Select(data, x => x % 10 == 0).ToList();

            Assert.That(10, Is.EqualTo(result.Count));
            CollectionAssert.Contains(result, 10);
            CollectionAssert.Contains(result, 100);
        }
    }
}
```