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.

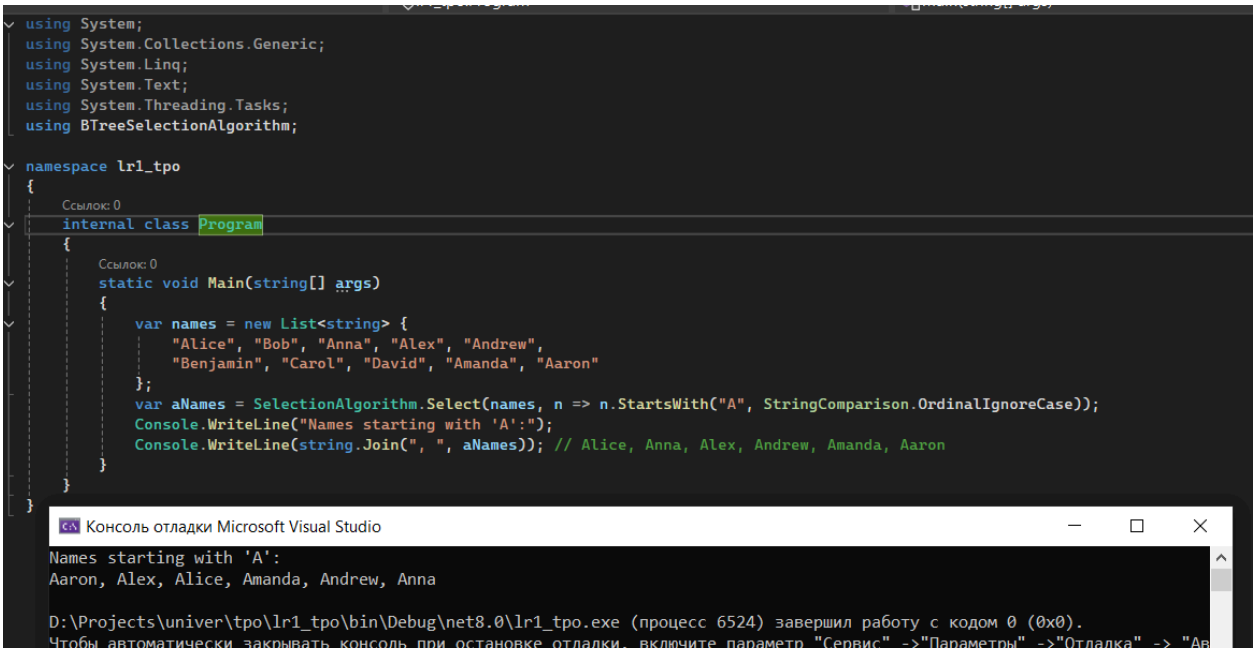
Вариант 17:

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	

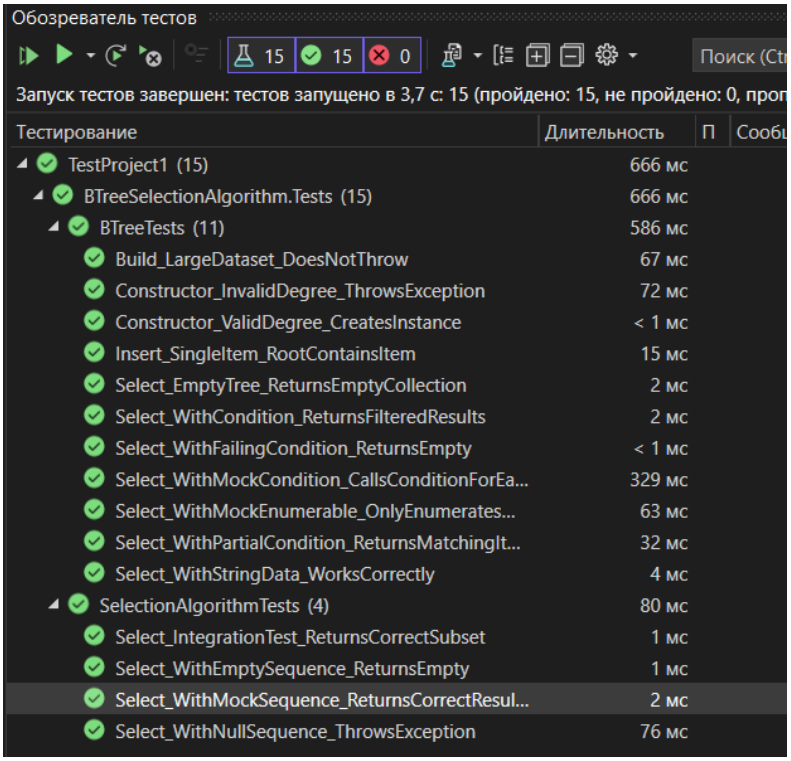
Работа выполнена на языке C# 12 в Visual Studio 2022 с использование NUnit, Moq, Coverlet.

```
<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>
```

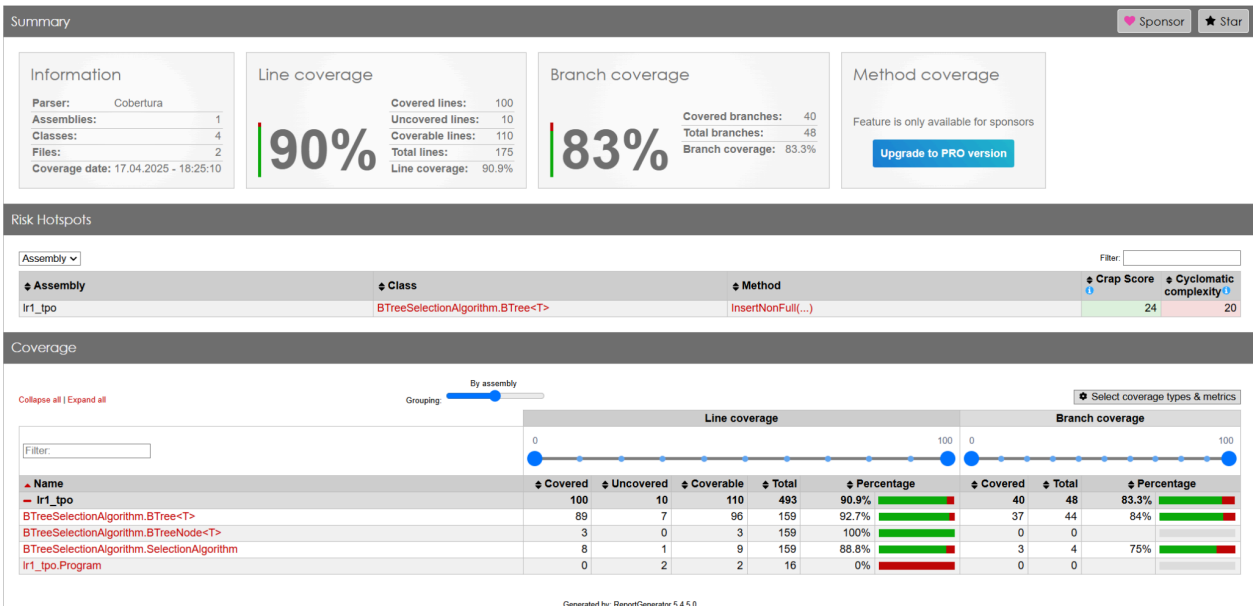
Результат работы алгоритма:



Результат тестирования:



Покрытие кода тестами:



Код алгоритма:

Btree.cs

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

namespace BTreeSelectionAlgorithm
{
    // Узел B-дерева
    public class BTreeNode<T> where T : IComparable<T>
    {
        public List<T> Keys { get; } = new List<T>();
        public List<BTreeNode<T>> Children { get; } = new List<BTreeNode<T>>();
        public bool IsLeaf => Children.Count == 0;
    }

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

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

// 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)
{
    var root = _root;
    if (root.Keys.Count == (2 * _degree) - 1)
    {
        var newRoot = new BTreeNode<T>();
        newRoot.Children.Add(root);
        SplitChild(newRoot, 0);
        _root = newRoot;
        InsertNonFull(newRoot, key);
    }
    else
```

```
    {
        InsertNonFull(root, key);
    }
}

// Поиск места для вставки
private void InsertNonFull(BTreeNode<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);
            if (key.CompareTo(node.Keys[i]) > 0)
            {
                i++;
            }
        }
        InsertNonFull(node.Children[i], key);
```

```
    }
}

// Разделение потомка на два узла
private void SplitChild(BTreeNode<T> parentNode, int childIndex)
{
    var child = parentNode.Children[childIndex];
    var newNode = new BTreeNode<T>();
    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(BTreeNode<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));
```

```
            const int degree = 100;
            var btree = new BTree<T>(degree);
            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 = 3;
        private BTree<int> _btree;

        [SetUp]
        public void Setup()
        {
            _btree = new BTree<int>(Degree);
        }

        [Test]
        public void Constructor_ValidDegree_CreatesInstance()
        {
            Assert.DoesNotThrow(() => new BTree<int>(2));
```

```
}

[Test]
public void Constructor_InvalidDegree_ThrowsException()
{
    Assert.Throws<ArgumentException>(() => new BTree<int>(1));
}

[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_WithCondition_ReturnsFilteredResults()
{
    var data = new[] { 10, 20, 30, 40, 50 };
    _btree.Build(data);

    var result = _btree.Select(x => x > 25).ToList();

    Assert.That(3, Is.EqualTo(result.Count));
    CollectionAssert.Contains(result, 30);
    CollectionAssert.Contains(result, 40);
    CollectionAssert.Contains(result, 50);
}

[Test]
public void Select_EmptyTree_ReturnsEmptyCollection()
```

```
{
    var result = _btree.Select(x => true);
    CollectionAssert.IsEmpty(result);
}

[Test]
public void Build_LargeDataset_DoesNotThrow()
{
    var largeData = Enumerable.Range(1, 10000);
    Assert.DoesNotThrow(() => _btree.Build(largeData));
}

[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 Select_WithPartialCondition_ReturnsMatchingItems()
{
    var data = new[] { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
    _btree.Build(data);

    var result = _btree.Select(x => x % 2 == 0).ToList();

    Assert.That(5, Is.EqualTo(result.Count));
    CollectionAssert.AreEquivalent(new[] { 2, 4, 6, 8, 10 }, result);
}

[Test]
public void Select_WithStringData_WorksCorrectly()
{
    var stringTree = new BTree<string>(Degree);
    stringTree.Build(new[] { "apple", "banana", "cherry" });

    var result = stringTree.Select(x => x.StartsWith("a")).ToList();

    Assert.That(1, Is.EqualTo(result.Count));
    StringAssert.AreEqualIgnoringCase("apple", result[0]);
}

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

    var tree = new BTree<int>(Degree);
```

```
        tree.Build(mockSequence.Object);

        mockSequence.Verify(x => x.GetEnumerator(), Times.Once);
    }
}

[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);
        }
    }
}
```