МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Ижевский государственный технический университет имени М. Т. Калашникова»
Факультет «Информационные технологии»
Кафедра «Программное обеспечение»

ОТЧЕТ

По лабораторной работе №1

По дисциплине «Тестирование ПО»

Выполнил
студент гр. Б22-191-1з Валеев Д.В.

Проверила: Старыгина Е. В.

1. ЗАДАНИЕ

Дан массив действительных чисел a0,...,an-1. Если в данном массиве есть хотя бы один член, меньший чем -1, то все отрицательные члены заменить их кубами.

На входе функции: массив действительных чисел, длина массива может быть любой

На выходе функции: массив действительных чисел массив ошибок при выполнении функции

2. БЛОК-СХЕМА, ОПИСАНИЕ АЛГОРИТМА.

Блок-схема алгоритма

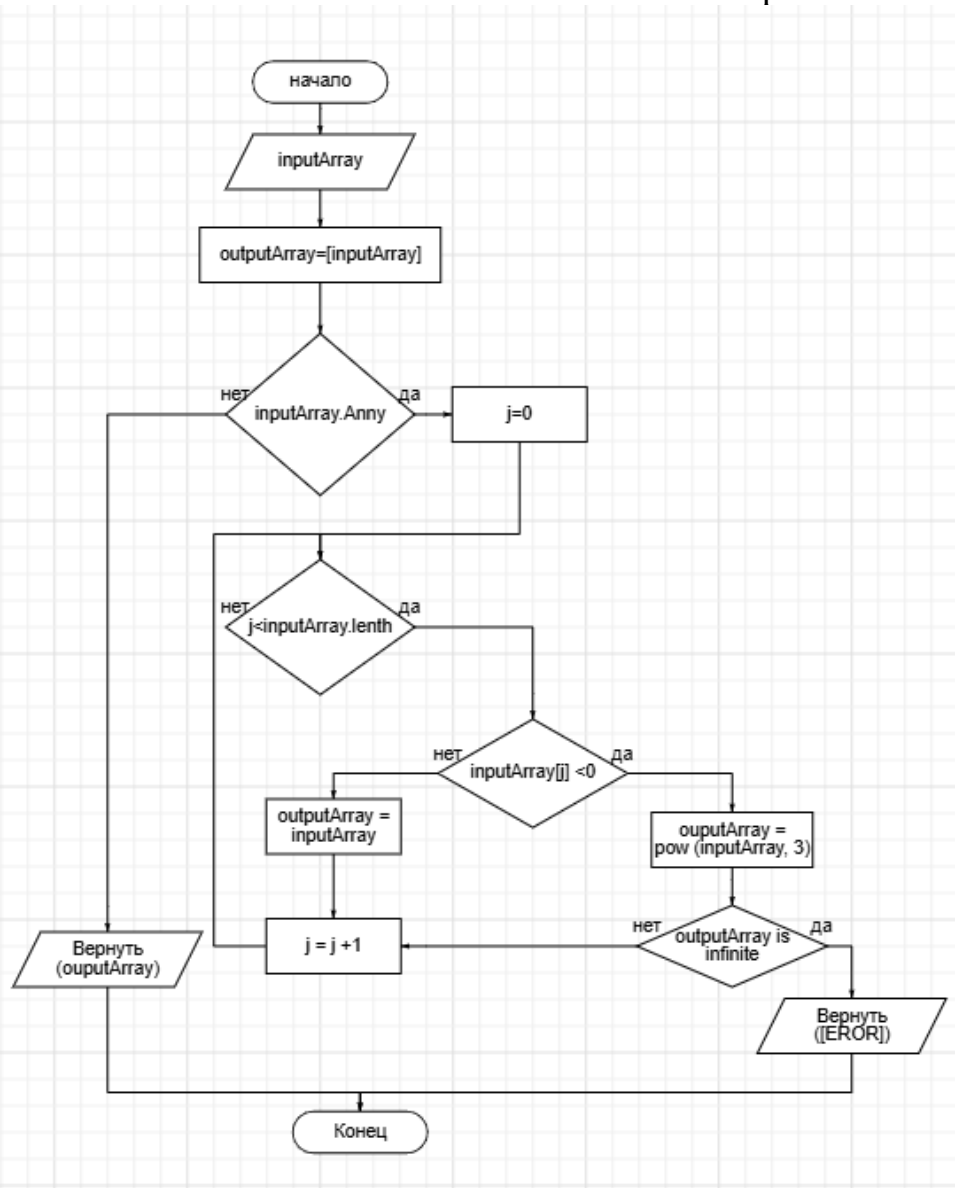

Рис. 1

Алгоритм проверяет нахождение элемента меньше -1 в массиве. Если его нет, то копирует оригинальный массив и возвращает его. Если элемент

есть меньше -1, то проходится по всем элементам массива. Если элемент отрицательный возводит его в куб, если нет, то записывает его без изменений.

3. ТЕСТИРОВАНИЕ БАЗОВОГО ПУТИ.

3.1. Потоковый граф

Потоковый граф алгоритма

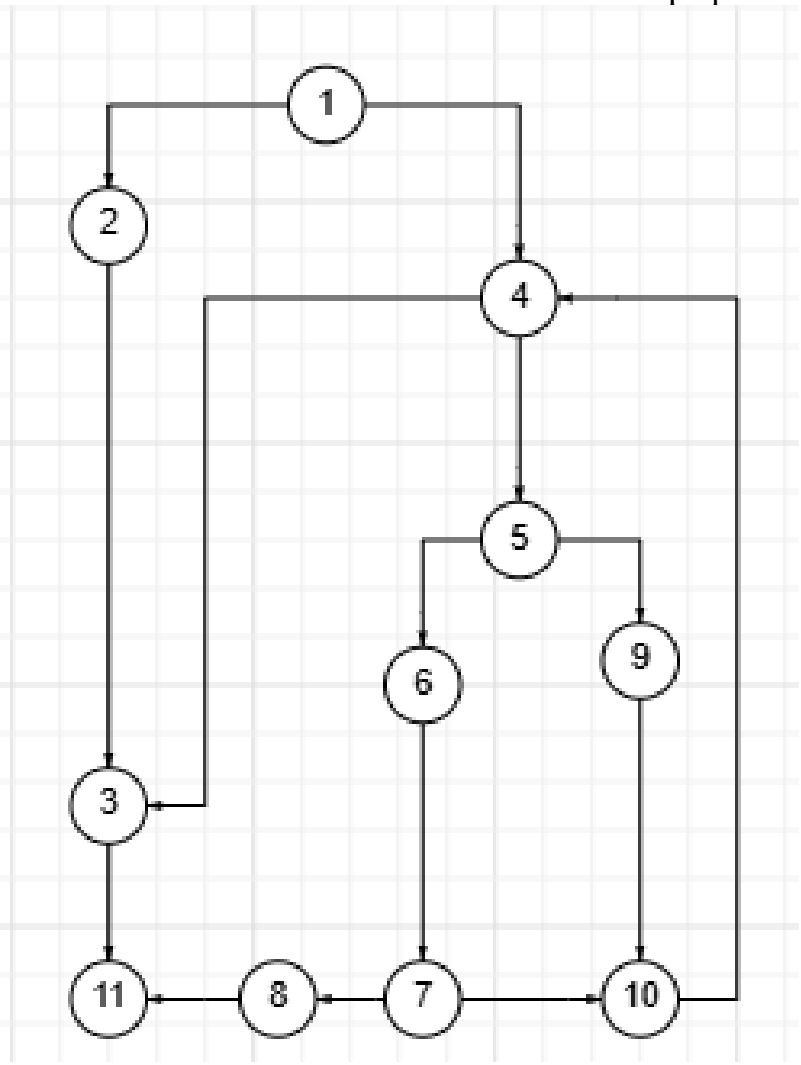

Рис. 2

3.2. Цикломатическая сложность.

Цикломатическая сложность вычисляется одним из трех способов:

1. цикломатическая сложность равна количеству регионов потокового графа;
2. цикломатическая сложность определяется по формуле V(G)-E-N+2, где E — количество дуг, N — количество узлов потокового графа;

3. цикломатическая сложность формируется по выражению V(G) =p+ 1, где p — количество предикатных узлов в потоковом графе G.

Вычисление цикломатическую сложность графа:

1) потоковый граф имеет 5 регионов;

2) V(G) = 14 дуг - 11 узлов + 2 = 5;

3) V(G) = 4 предикатных узла + 1 = 4.

Таким образом, цикломатическая сложность потокового графа равна 5.

3.3. Базовое множество независимых линейных путей.

Путь 1: 1-2-3-11

Путь 2: 1-4-3-11

Путь 3: 1-4-5-6-7-8-11

Путь 4: 1-4-5-6-7-10-4-11

Путь 5: 1-4-5-6-7-10-4-5-9-10-4-11

3.4. Тестовые варианты.

1. Тестовый вариант для пути 1 ТВ1:

   ИД: inputArray не содержит значение меньше -1.

   ОЖ.РЕЗ.: inputArray без изменений и пустой массив ошибок.

2. Тестовый вариант для пути 2 ТВ2:

   Данный вариант невозможен т.к. пустой inputArray не может содержать значение меньше -1.

3. Тестовый вариант для пути 3 ТВ3:

   ИД: inputArray содержит значение меньше -1 и один из элементов при возведении в куб дает -Infinity.

   ОЖ.РЕЗ.: пустой массив и ошибка о значении -Infinity.

4. Тестовый вариант для пути 4 ТВ4:

ИД: inputArray содержит значение меньше -1 и все элементы отрицательные.

ОЖ.РЕЗ.: все элементы списка будут возведены в куб.

5. Тестовый вариант для пути 5 ТВ5:

   ИД: inputArray содержит значение меньше -1 и есть элементы положительные элементы.

   ОЖ.РЕЗ.: отрицательные элементы будут возведены в куб, положительные останутся без изменений.

4. ТЕСТИРОВАНИЕ ПОТОКОВ ДАННЫХ.

4.1. Информационный граф.

Информационный граф алгоритма

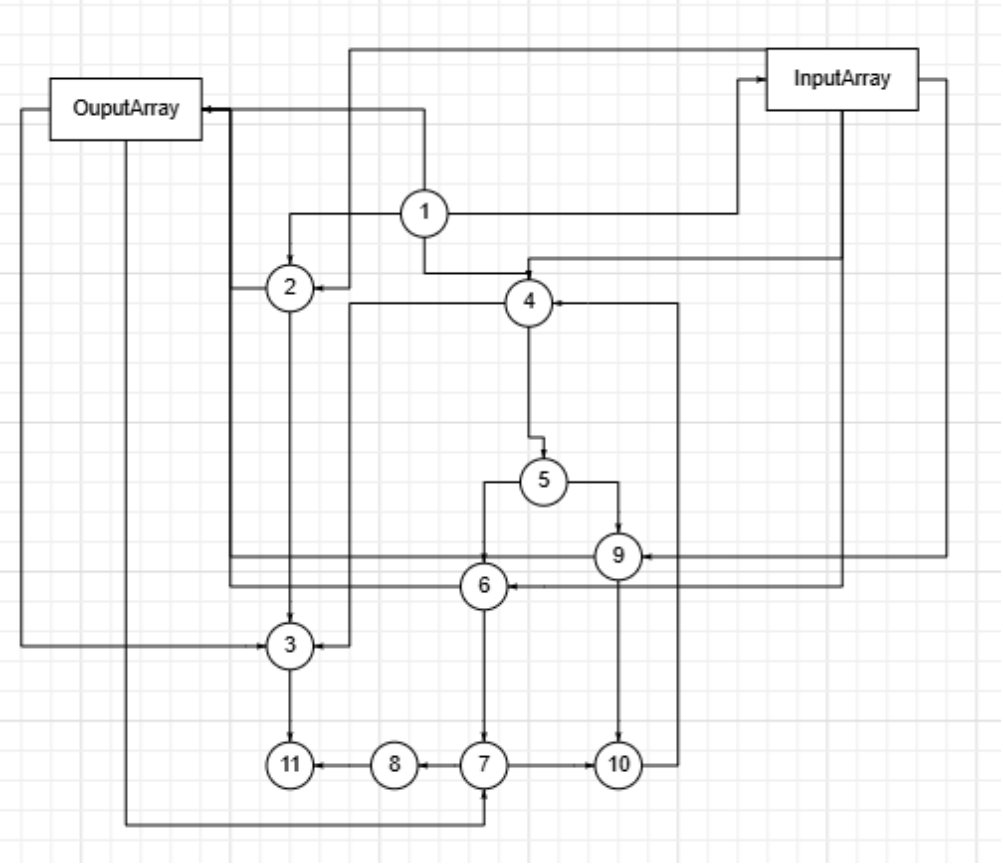

Рис. 3

4.2. Формирование полного набора DU-цепочек.

В данном алгоритме существуют следующие DU-цепочки:
[inputArray,1,2],[inputArray,1,4][inputArray,1,5][inputArray,1,6][inputArray,1,9],[outputArray,1,3],[outputArray,2,3] [outputArray,6,7], [outputArray,6,3], [outputArray,9,3]

4.3. Формирование полного набора отрезков путей в управляющем графе.

Отрезки пути для inputArray:

1. 1-2
2. 1-4
3. 1-4-5

4. 1-4-5-6
5. 1-4-5-9

Отрезки пути для inputArray:

1. 1-2-3
2. 2-3
3. 6-7-10-4-3
4. 9-10-4-3

4.4. Построение маршрутов.

1. Путь 1: 1-2-3-11
2. Путь 2: 1-4-3-11
3. Путь 4: 1-4-5-6-7-10-4-11
4. Путь 5: 1-4-5-6-7-10-4-5-9-10-4-11

5. ОБЛАСТИ ЭКВИВАЛЕНТНОСТИ.

При определении областей эквивалентности руководствуются следующими правилами:

1. Тестирующая последовательность может состоять из одного элемента. Обычно считается, что последовательности состоят из нескольких элементов и программисты иногда закладывают такое представление в свои программы. Следовательно если ввести последовательность из одного элемента, программа может сработать неправильно.
2. Следует использовать в разных тестах различные последовательности, содержащие разное количество элементов. Это уменьшает вероятность того, что программа имеющая дефекты, случайно выдает правильные результаты в силу некоторых случайных свойств входных данных.
3. Следует использовать тестирующие последовательности, в которых ключевой элемент является первым, средним и последним элементом последовательности. Такой метод помогает выявить проблемы на границах областей эквивалентности.

Таблица 1. Области эквивалентности для программы поиска

№	Последовательность	Ключевой элемент	Тип данных
1	Нет элементов	Нет в последовательности	нет
2	Один элемент	Есть в последовательности	min < el < max
3	Один элемент	Есть в последовательности	el = min
4	Один элемент	Нет в последовательности	min < el < max
5	Один элемент	Нет в последовательности	el = max

6	Несколько элементов	Первый элемент последовательности	min < el < max
7	Несколько элементов	Первый элемент последовательности	el = min
8	Несколько элементов	Последний элемент последовательности	min < el < max
9	Несколько элементов	Последний элемент последовательности	el = min
10	Несколько элементов	Средний элемент последовательности	min < el < max
11	Несколько элементов	Средний элемент последовательности	el = min
12	Несколько элементов	Нет в последовательности	min < el < max
13	Несколько элементов	Нет в последовательности	el = max

6. КОНТРОЛЬНЫЙ ПРИМЕР.

Для отображения прохода по тестовому пути использовались инструменты встроенные в Visual Studio. Код по которому осуществлен проход подсвечивается синим цветом, белым - если прохода не было.

1. Тестовый вариант 1.

Пример работы тестового варианта 1.

```
Тестовый вариант: Вариант 1
Набор: Набор 2
Путь в потоковом графе: 1-2-3-11
Исходные данные: -1
Ожидаемые результаты: -1
Результат, полученный программой: -1
Вывод по проверке результата теста: Тест пройден
```

Рис. 4

2. Тестовый вариант 3.

Пример работы тестового варианта 2.

```
Тестовый вариант: Вариант 2
Набор: Набор 3
Путь в потоковом графе: 1-4-5-6-7-8-11
Исходные данные: 94875772
Ожидаемые результаты:
Результат, полученный программой:
Вывод по проверке результата теста: Тест пройден
```

Рис. 5

3. Тестовый вариант 3 .

Пример работы тестового варианта 3.

```
Тестовый вариант: Вариант 3
Набор: Набор 3
Путь в потоковом графе: 1-4-5-6-7-10-4-11
Исходные данные: -1
Ожидаемые результаты: -1
Результат, полученный программой: -1
Вывод по проверке результата теста: Тест пройден
```

Рис. 6

7. ПРОТОКОЛ ТЕСТИРОВАНИЯ

Таблица 2. Протокол тестирования

№	Вариант	Набор	Путь	Длина массива	Входные данные	Ожидаемый результат	Фактический результат	Вывод по тесту
1	1	1	1-2-3-11	0	Пустой массив	Массив без изменений	Массив без изменений	Успех
2	1	2	1-2-3-11	1	Массив с одним элементом большим или равным -1	Массив без изменений	Массив без изменений	Успех
3	1	3	1-2-3-11	93	Массив с несколькими элементами большии или равным -1	Массив без изменений	Массив без изменений	Успех
4	1	4	1-2-3-11	186	Массив с несколькими элементами большии или равным -1 и элемент находящийся на макс. границе	Массив без изменений	Массив без изменений	Успех
5	3	1	1-4-5-6-7-8-11	1	Массив с элементом, который при возведении в куб дает -Infinity	Пустой массив и ошибка	Пустой массив и ошибка	Успех
6	3	2	1-4-5-6-7-8-11	279	Массив с несколькими элементами, элемент, который при возведении в куб дает -Infinity в начале	Пустой массив и ошибка	Пустой массив и ошибка	Успех
7	3	3	1-4-5-6-7-8-11	372	Массив с несколькими элементами, элемент, который при возведении в куб дает -Infinity в середине	Пустой массив и ошибка	Пустой массив и ошибка	Успех
8	3	4	1-4-5-6-7-8-11	465	Массив с несколькими элементами, элемент, который	Пустой массив и ошибка	Пустой массив и ошибка	Успех

					при возведении в куб дает -Infinity в конце			
9	4	1	1-4-5-6-7-10-4-11	1	Массив с одним элементом меньше -1	Отрицательные элементы возведены в куб	Отрицательные элементы возведены в куб	Успех
10	4	2	1-4-5-6-7-10-4-11	1	Массив с одним элементом меньше -1, который при возведении в куб возвращает максимальное число для double	Отрицательные элементы возведены в куб	Отрицательные элементы возведены в куб	Успех
11	4	3	1-4-5-6-7-10-4-11	1	Массив с одним элементом меньше -1, который имеет минимальную разницу с -1	Отрицательные элементы возведены в куб	Отрицательные элементы возведены в куб	Успех
12	4	4	1-4-5-6-7-10-4-11	558	Массив с несколькими отрицательными элементами, элемент меньше -1 находится в начале	Отрицательные элементы возведены в куб	Отрицательные элементы возведены в куб	Успех
13	4	5	1-4-5-6-7-10-4-11	651	Массив с несколькими отрицательными элементами, элемент меньше -1 находится в середине	Отрицательные элементы возведены в куб	Отрицательные элементы возведены в куб	Успех
14	4	6	1-4-5-6-7-10-4-11	744	Массив с несколькими отрицательными элементами, элемент меньше -1 находится в конце	Отрицательные элементы возведены в куб	Отрицательные элементы возведены в куб	Успех
15	5	1	1-4-5-6-7-10-4-5-9-10-4-11	2	Массив с одним элементом меньше -1и одним положительным элеметом	Отрицательные элементы возведены в куб, положител	Отрицательные элементы возведены в куб, положител	Успех

						ьные без изменений	ьные без изменений	
16	5	2	1-4-5-6-7-10-4-5-9-10-4-11	2	Массив с одним элементом меньше -1и 0	Отрицательные элементы возведены в куб, положительные без изменений	Отрицательные элементы возведены в куб, положительные без изменений	Успех
17	5	3	1-4-5-6-7-10-4-5-9-10-4-11	2	Массив с одним элементом меньше -1и элементом находящийся на макс. границе	Отрицательные элементы возведены в куб, положительные без изменений	Отрицательные элементы возведены в куб, положительные без изменений	Успех
18	5	4	1-4-5-6-7-10-4-5-9-10-4-11	837	Массив с несколькими положительными элементами и элемент меньше -1 находится в начале	Отрицательные элементы возведены в куб, положительные без изменений	Отрицательные элементы возведены в куб, положительные без изменений	Успех
19	5	5	1-4-5-6-7-10-4-5-9-10-4-11	930	Массив с несколькими положительными элементами и элемент меньше -1 находится в середине	Отрицательные элементы возведены в куб, положительные без изменений	Отрицательные элементы возведены в куб, положительные без изменений	Успех
20	5	6	1-4-5-6-7-10-4-5-9-10-4-11	1023	Массив с несколькими положительными элементами и элемент меньше -1 находится в конце	Отрицательные элементы возведены в куб, положительные без изменений	Отрицательные элементы возведены в куб, положительные без изменений	Успех

8. ТЕКСТ ПРОГРАММЫ.

1. test-os.csproj

```
<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>
    <OutputType>Exe</OutputType>
            <TargetFramework>net8.0</TargetFramework>
    <RootNamespace>test_os</RootNamespace>
    <ImplicitUsings>enable</ImplicitUsings>
    <Nullable>enable</Nullable>
  </PropertyGroup>

</Project>
```

2. Program.cs

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

namespace MainProgram
{
    public class Program
    {
        static void Main()
        {
            double[] inputArray = { 1.5, -1.0, 3.0, -0.5, -0.2 };
            double[] outputArray;
            string[] errors;

            (outputArray, errors) = ProcessArray(inputArray);
            if (errors.Length != 0)
            {
                Console.WriteLine("\nОшибки:");
                foreach (var error in errors)
                {
                    Console.WriteLine(error);
                }
                return;
            }
        }

        static public (double[] outputArray, string[] errors) ProcessArray(double[] inputArray)
        {
            var outputArray = new double[inputArray.Length];

            if (inputArray.Any(e => e < -1))
            {
                for (int i = 0; i < inputArray.Length; i++)
                {
                    if (inputArray[i] < 0)
                    {
                        outputArray[i] = Math.Pow(inputArray[i], 3);
                        if (double.IsInfinity(outputArray[i]))
```

```
                    {
                        return (new double[0], new string[] { "Error: -Infinity" });
                    }
                }
                else
                {
                    outputArray[i] = inputArray[i];
                }
            }
        }
        else
        {
            Array.Copy(inputArray, outputArray, inputArray.Length);
        }

        return (outputArray, new string[0]);
    }

  }
}
```

3. tests.csproj

```
<Project Sdk="Microsoft.NET.Sdk">

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

  <IsPackable>false</IsPackable>
  <IsTestProject>true</IsTestProject>
        <StartupObject>GenerateTestData.Test</StartupObject>
 </PropertyGroup>

 <ItemGroup>
  <PackageReference Include="coverlet.collector" Version="6.0.0" />
  <PackageReference Include="Microsoft.NET.Test.Sdk" Version="17.8.0" />
  <PackageReference Include="xunit" Version="2.5.3" />
  <PackageReference Include="xunit.runner.visualstudio" Version="2.5.3" />
 </ItemGroup>

 <ItemGroup>
  <ProjectReference Include="..\test-os\test-os.csproj" />
 </ItemGroup>

 <ItemGroup>
  <Using Include="Xunit" />
 </ItemGroup>

</Project>
```

4. Models.cs

```
public class TestCase{
```

```
    public TestCase(
        double[] InputArray,
        double[] ExpectedOutput,
        string[] ExpectedErrors,
        string Variant,
        string Path,
        string Set
    ){
        this.InputArray = InputArray;
        this.ExpectedOutput = ExpectedOutput;
        this.ExpectedErrors = ExpectedErrors;
        this.Variant = Variant;
        this.Set = Set;
        this.Path = Path;
    }

    public double[] InputArray { get; set; }
    public double[] ExpectedOutput { get; set; }
    public string[] ExpectedErrors { get; set; }
    public string Variant { get; set; }
    public string Set { get; set; }
    public string Path { get; set; }
}

    public class TestResult
    {

        public TestResult(
            string Variant,
            string Set,
            string Path,
            double[] Input,
            double[] ExpectedOutput,
            string[] ExpectedErrors,
            double[] ActualOutput,
            string[] ActualErrors
        ){
            this.Variant = Variant;
            this.Set = Set;
            this.Path = Path;
            this.Input = Input;
            this.ExpectedOutput = ExpectedOutput;
            this.ExpectedErrors = ExpectedErrors;
            this.ActualOutput = ActualOutput;
            this.ActualErrors = ActualErrors;
        }
        public string Variant { get; set; }
        public string Set { get; set; }
        public string Path { get; set; }
        public double[] Input { get; set; }
        public double[] ExpectedOutput { get; set; }
        public string[] ExpectedErrors { get; set; }
        public double[] ActualOutput { get; set; }
        public string[] ActualErrors { get; set; }

        public bool IsPassed =>
            ExpectedOutput.SequenceEqual(ActualOutput) &&
            ExpectedErrors.SequenceEqual(ActualErrors);
```

```
    public override string ToString()
    {

        return string.Join("\n", [
            $"Тестовый вариант: {Variant}",
            $"Набор: {Set}",
            $"Путь в потоковом графе: {Path}",
            $"Исходные данные: {string.Join(" ", Input)}",
            $"Ожидаемые результаты: {string.Join(" ", ExpectedOutput)}",
            $"Результат, полученный программой: {string.Join(" ", ActualOutput)}",
            $"Вывод по проверке результата теста: {(IsPassed ? "Тест пройден" : "Тест не пройден")}"
        ]);
    }
}
```

4. GenerateTestData.cs

```
using System.Globalization;
using System.Text;

namespace GenerateTestData
{
    public class Test
    {
        static void Main()
        {
            Gen();
        }
        static public void Gen()
        {
            string path =
            @"C:\Users\dmitr\OneDrive\Desktop\gfvjhkl,\tests\test_cases.t
            xt";
             string path = Path.Combine(currentDirectory, "test_cases.txt");

            using var writer = new StreamWriter(path, false, Encoding.UTF8);


            Variantgenerator[] array = [
                new GenerateVariant1(),
                new GenerateVariant3(),
                new GenerateVariant4(),
                new GenerateVariant5(),
            ];

            foreach (var item in array)
            {
                var el = item.Generate();

                foreach (var testCase in el)
                {
                    string input = string.Join(" ", testCase.InputArray.Select(x => x.ToString("G17",
CultureInfo.InvariantCulture)));
                    string output = string.Join(" ", testCase.ExpectedOutput.Select(x => x.ToString("G17",
CultureInfo.InvariantCulture)));
                    string error = string.Join(" ", testCase.ExpectedErrors);
                    writer.WriteLine($"{input},{testCase.Variant},{testCase.Set},{testCase.Path}");
                    writer.WriteLine($"{output},{error},{testCase.Variant},{testCase.Set},{testCase.Path}");
                }
```

```
        }
    }
}

abstract class Variantgenerator
{
    public abstract TestCase[] Generate();

    public abstract string Variant { get; }

    public abstract string Path { get; }

    public IEnumerable<double> GenerateArray(int length, double min, double max)
    {
        if (double.IsNaN(min) || double.IsNaN(max))
            throw new ArgumentException("min и max должны быть валидными");

        if (min > max)
            throw new ArgumentException("min больше чем max");

        var rnd = new Random();

        for (int i = 0; i < length; i++)
        {

            yield return NextDouble(rnd, min, max);
        }
    }

    static double NextDouble(Random random, double min, double max)
    {
        if(max > int.MaxValue){
            max = int.MaxValue;
        }

        if(min < int.MinValue){
            max = int.MinValue;
        }

        double range = max - min;



        return random.NextDouble() * (max - min) + min;
    }
}

class ValueCounter
{
    private const int step = 93;

    static private int counter = 1;

    static public int next()
    {
        return step * counter++;
```

```
    }
}
class GenerateVariant1 : Variantgenerator
{

    public override string Variant
    {
        get { return "Вариант 1"; }
    }

    public override string Path
    {
        get { return "1-2-3-11"; }
    }

    public override TestCase[] Generate()
    {


        var result = new TestCase[4];

        result[0] = new TestCase(
                InputArray: new double[0],
                ExpectedOutput: new double[0],
                ExpectedErrors: new string[0],
                Variant: Variant,
                Path: Path,
                Set: "Набор 1"

            );

        result[1] = new TestCase(
            InputArray: [-1],
            ExpectedOutput: [-1],
            ExpectedErrors: new string[0],
            Variant: Variant,
            Path: Path,
            Set: "Набор 2"
        );


        var input1 = GenerateArray(ValueCounter.next(), -1, double.MaxValue).ToArray();

        result[2] = new TestCase(
                InputArray: input1,
                ExpectedOutput: input1,
                ExpectedErrors: new string[0],
                Variant: Variant,
                Path: Path,
                Set: "Набор 3"
            );

        var input2 = GenerateArray(ValueCounter.next(), -1, double.MaxValue).ToArray();

        input2[0] = double.MaxValue;

        result[3] = new TestCase(
                InputArray: input2,
                ExpectedOutput: input2,
```

```
            ExpectedErrors: new string[0],
            Variant: Variant,
            Path: Path,
            Set: "Набор 4"
        );

    return result;
  }
}

class GenerateVariant3 : Variantgenerator
{

  public override string Variant
  {
    get { return "Вариант 3"; }
  }

  public override string Path
  {
    get { return "1-4-5-6-7-8-11"; }
  }
  public override TestCase[] Generate()
  {
    var result = new TestCase[4];

    result[0] = new TestCase(
      InputArray: [double.MinValue],
      ExpectedOutput: new double[0],
      ExpectedErrors: ["Error: -Infinity"],
      Variant: Variant,
      Path: Path,
      Set: "Набор 1"
    );

    var input1 = GenerateArray(ValueCounter.next(), -1, double.MaxValue).ToArray();

    input1[0] = double.MinValue;

    result[1] = new TestCase(
      InputArray: input1,
      ExpectedOutput: new double[0],
      ExpectedErrors: ["Error: -Infinity"],
      Variant: Variant,
      Path: Path,
      Set: "Набор 2"
    );


    var input2 = GenerateArray(ValueCounter.next(), -1, double.MaxValue).ToArray();

    input2[input2.Length / 2] = double.MinValue;

    result[2] = new TestCase(
      InputArray: input2,
      ExpectedOutput: new double[0],
      ExpectedErrors: ["Error: -Infinity"],
      Variant: Variant,
      Path: Path,
```

```
            Set: "Набор 3"
        );

        var input3 = GenerateArray(ValueCounter.next(), -1, double.MaxValue).ToArray();

        input3[input3.Length - 1] = double.MinValue;

        result[3] = new TestCase(
            InputArray: input3,
            ExpectedOutput: new double[0],
            ExpectedErrors: ["Error: -Infinity"],
            Variant: Variant,
            Path: Path,
            Set: "Набор 4"
        );
        return result;
    }
}

class GenerateVariant4 : Variantgenerator
{
    public override string Variant
    {
        get { return "Вариант 4"; }
    }

    public override string Path
    {
        get { return "1-4-5-6-7-10-4-11"; }
    }
    public override TestCase[] Generate()
    {
        var result = new TestCase[6];

        result[0] = new TestCase(
                InputArray: [-3],
                ExpectedOutput: [-27],
                ExpectedErrors: new string[0],
                Variant: Variant,
                Path: Path,
                Set: "Набор 1"
            );

        result[1] = new TestCase(
                InputArray: [-Math.Pow(-double.MinValue, 1.0 / 3)],
                ExpectedOutput: [Math.Pow(-Math.Pow(-double.MinValue, 1.0 / 3), 3)],
                ExpectedErrors: new string[0],
                Variant: Variant,
                Path: Path,
                Set: "Набор 2"
            );

        result[2] = new TestCase(
                InputArray: [-1 - double.Epsilon],
                ExpectedOutput: [Math.Pow(-1 - double.Epsilon, 3)],
                ExpectedErrors: new string[0],
                Variant: Variant,
                Path: Path,
                Set: "Набор 3"
```

```
        );

    var input1 = GenerateArray(ValueCounter.next(), -1, 0).ToArray();

    input1[0] = -4;
    var output1 = pow(input1);

    result[3] = new TestCase(
        InputArray: input1,
        ExpectedOutput: output1,
        ExpectedErrors: new string[0],
        Variant: Variant,
        Path: Path,
        Set: "Набор 4"
      );

    var input2 = GenerateArray(ValueCounter.next(), -1, 0).ToArray();
    input2[input2.Length / 2] = -5;

    var output2 = pow(input2);

    result[4] = new TestCase(
        InputArray: input2,
        ExpectedOutput: output2,
        ExpectedErrors: new string[0],
        Variant: Variant,
        Path: Path,
        Set: "Набор 5"
      );

    var input3 = GenerateArray(ValueCounter.next(), -1, 0).ToArray();
    input3[input3.Length - 1] = -2;

    var output3 = pow(input3);

    result[5] = new TestCase(
        InputArray: input3,
        ExpectedOutput: output3,
        ExpectedErrors: new string[0],
        Variant: Variant,
        Path: Path,
        Set: "Набор 6"
      );
    return result;
  }

  private double[] pow(double[] array)
  {
    var result = new double[array.Length];
    for (int i = 0; i < array.Length; i++)
    {
      result[i] = Math.Pow(array[i], 3);
    }
    return result;
  }
}
class GenerateVariant5 : Variantgenerator
{
```

```
public override string Variant
{
   get { return "Вариант 5"; }
}

public override string Path
{
   get { return "1-4-5-6-7-10-4-5-9-10-4-11"; }
}
public override TestCase[] Generate()
{
   var result = new TestCase[6];

   result[0] = new TestCase(
      InputArray: [-3, 25],
      ExpectedOutput: [-27, 25],
      ExpectedErrors: new string[0],
      Variant: Variant,
      Path: Path,
      Set: "Набор 1"
   );

   result[1] = new TestCase(
      InputArray: [-3, 0],
      ExpectedOutput: [-27, 0],
      ExpectedErrors: new string[0],
      Variant: Variant,
      Path: Path,
      Set: "Набор 2"
   );

   result[2] = new TestCase(
      InputArray: [-3, double.MaxValue],
      ExpectedOutput: [-27, double.MaxValue],
      ExpectedErrors: new string[0],
      Variant: Variant,
      Path: Path,
      Set: "Набор 3"
   );

   var input1 = GenerateArray(ValueCounter.next(), 0, double.MaxValue).ToArray();
   var output1 = new double[input1.Length];
   Array.Copy(input1, output1, input1.Length);
   input1[0] = -4;
   output1[0] = -64;

   result[3] = new TestCase(
      InputArray: input1,
      ExpectedOutput: output1,
      ExpectedErrors: new string[0],
      Variant: Variant,
      Path: Path,
      Set: "Набор 4"
   );

   var input2 = GenerateArray(ValueCounter.next(), 0, double.MaxValue).ToArray();
   var output2 = new double[input2.Length];
   Array.Copy(input2, output2, input2.Length);
   input2[input2.Length / 2] = -5;
```

```
            output2[input2.Length / 2] = -125;

            result[4] = new TestCase(
                InputArray: input2,
                ExpectedOutput: output2,
                ExpectedErrors: new string[0],
                Variant: Variant,
                Path: Path,
                Set: "Набор 5"
            );

            var input3 = GenerateArray(ValueCounter.next(), 0, double.MaxValue).ToArray();
            var output3 = new double[input3.Length];
            Array.Copy(input3, output3, input3.Length);
            input3[input3.Length - 1] = -2;
            output3[input3.Length - 1] = -8;

            result[5] = new TestCase(
                InputArray: input3,
                ExpectedOutput: output3,
                ExpectedErrors: new string[0],
                Variant: Variant,
                Path: Path,
                Set: "Набор 6"
            );
            return result;
        }
    }
}
```

5. UnitTest1.cs

```
using MainProgram;
using System.Text;
using System.Globalization;
using System.Diagnostics;

namespace tests
{
    public class TestRunner: IDisposable
    {

        private static string TestFilePath {
            get
            {
                string currentDirectory = Directory.GetCurrentDirectory();
                return Path.Combine(currentDirectory, "test_cases.txt");
            }
        }
        private static string ResultsFilePath {
            get
            {
                string currentDirectory = Directory.GetCurrentDirectory();
                return Path.Combine(currentDirectory, "test_results.txt");
            }
        }

        public TestRunner()
        {
```

```
    this.writer = new StreamWriter(ResultsFilePath, true, Encoding.UTF8);
}

private readonly StreamWriter writer;

 public void Dispose()
{
    writer.Dispose();
    GC.SuppressFinalize(this);
}

[Theory]
[MemberData(nameof(ReadTestCases))]
public void RunTests(TestCase testCase)
{
    var (outputArray, errors) = Program.ProcessArray(testCase.InputArray);

    Assert.Equal(testCase.ExpectedErrors, errors);
    Assert.Equal(testCase.ExpectedOutput, outputArray);

    var testResult = new TestResult(
        Variant: testCase.Variant,
        Set: testCase.Set,
        Path: testCase.Path,
        Input: testCase.InputArray,
        ExpectedOutput: testCase.ExpectedOutput,
        ExpectedErrors: testCase.ExpectedErrors,
        ActualOutput: outputArray,
        ActualErrors: errors
    );
    writer.WriteLine(testResult.ToString());
}

public static IEnumerable<object[]> ReadTestCases()
{
    if (File.Exists(ResultsFilePath))
    {
        File.Delete(ResultsFilePath);
    }

    using var reader = new StreamReader(TestFilePath, Encoding.UTF8);
    string? line;

    while((line = reader.ReadLine()) != null){
        var inputLine = line.Split(',');
        if(inputLine.Length == 0) break;

        var outputLine = reader.ReadLine()!.Split(',');
        yield return [
            new TestCase(
                Variant: inputLine[1].Trim(),
                Set: inputLine[2].Trim(),
                Path: inputLine[3].Trim(),
                InputArray: ParseDoubleArray(inputLine[0]),
                ExpectedOutput: ParseDoubleArray(outputLine[0]),
                ExpectedErrors: outputLine[1].Trim() == "" ? [] : [ outputLine[1].Trim() ]
            )
        ];
```

```
            }
        }

        private static double[] ParseDoubleArray(string input)
        {
            if (string.IsNullOrWhiteSpace(input)) return new double[0];
            return input.Split(' ', StringSplitOptions.RemoveEmptyEntries)
                        .Select(e => double.Parse( e, CultureInfo.InvariantCulture))
                        .ToArray();
        }
    }
}
```