Министерство науки и высшего образования Российской Федерации  
Федеральное государственное автономное образовательное учреждение высшего образования «Уральский федеральный университет имени первого Президента России Б.Н. Ельцина» (УрФУ)

Институт фундаментального образования

Департамент «школа бакалавриата»



ОТЧЕТ

отчет по лабораторной работе дисциплины «Алгоритмы и анализ

сложности»

**«Арифметика рациональных чисел»**

Студент: Чернышев Иван Антонович РИ-210911

Преподаватель: Трофимова Ольга Геннадиевна, Доцент, к.т.н.

Екатеринбург

2021

**Оглавление**

[Задание 2](#_Toc71567837)

[Теоретическая часть 2](#_Toc71567838)

[Реализация рационального числа и арифметических операций 3](#_Toc71567839)

[Алгоритм приведения дроби к несократимому виду 4](#_Toc71567840)

[Алгоритм преобразования рациональной дроби в периодическое число с плавающей точкой 5](#_Toc71567841)

[Алгоритм преобразования периодической дроби в обыкновенную 6](#_Toc71567842)

[Инструкция для пользователя 8](#_Toc71567843)

[Инструкция для программиста 8](#_Toc71567844)

[Тестирование 8](#_Toc71567845)

[Вывод 8](#_Toc71567846)

[Литература 9](#_Toc71567847)

[Приложение 10](#_Toc71567848)

[Тестирование 10](#_Toc71567849)

**Задание**

1. Рациональные числа состоят из числителя и знаменателя в виде неограниченных целых чисел.
2. Реализовать арифметические операции.
3. Реализовать алгоритм Евклида нахождения наибольшего общего делителя для приведения дроби к несократимому виду.
4. Реализовать алгоритм преобразования обыкновенной дроби m/n в периодическую a,b(c).
5. Реализовать преобразование периодической дроби в обыкновенную.

**Теоретическая часть**

Начнем с определения рациональных чисел, которое воспринимается наиболее естественно.

Рациональные числа – это числа, которые можно записать в виде положительной обыкновенной дроби, отрицательной обыкновенной дроби или числа нуль.

Из озвученного определения следует, что рациональным числом является:

* Любое натуральное число n. Действительно, можно представить любое натуральное число в виде обыкновенной дроби, например,
* Любое целое число, в частности, число нуль. В самом деле, любое целое число можно записать в виде либо положительной обыкновенной дроби, либо в виде отрицательной обыкновенной дроби, либо как нуль. Например, 26=26/1,

,

* Любая обыкновенная дробь (положительная или отрицательная). Это напрямую утверждается приведенным определением рациональных чисел.
* Любое смешанное число. Действительно, всегда можно представить смешанное число в виде неправильной обыкновенной дроби. Например:

и

* Любая конечная десятичная дробь или бесконечная периодическая дробь. Это так в силу того, что указанные десятичные дроби переводятся в обыкновенные дроби. К примеру:

Также понятно, что любая бесконечная непериодическая десятичная дробь НЕ является рациональным числом, так как она не может быть представлена в виде обыкновенной дроби.

Теперь мы можем с легкостью привести примеры рациональных чисел. Числа 4, 903, 100, 321 – это рациональные числа, так как они натуральные. Целые числа 58, −72, 0, −833 333 333 тоже являются примерами рациональных чисел. Обыкновенные дроби , – это тоже примеры рациональных чисел. Рациональными числами являются и числа , ,

Из приведенных примеров видно, что существуют и положительные и отрицательные рациональные числа, а рациональное число нуль не является ни положительным, ни отрицательным.

Таким образом, рациональными числами называют числа, которые можно записать в виде дроби , где z – целое число, а n – натуральное число

**Реализация рационального числа и арифметических операций**

Реализация рационального числа в коде представляет собой структуру, содержащую два поля: Числитель и знаменатель. Числитель и знаменатель являются объектами класса BigInteger, таким образом, точность вычислений не ограничена.

Арифметические операции, операции сравнения, операции создания и копирования чисел реализованы в целом наборе файлов и основываются на реализованных ранее в первой лабораторной работе методах для BigInteger. Все они представлены в проекте. Для основных функций были написаны юнит-тесты.

Также выдержки кода для основных методов приведены в приложении.

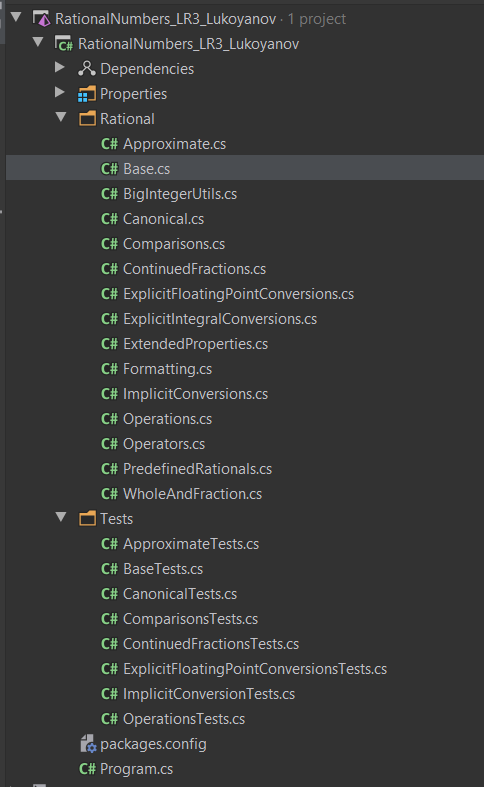


Рисунок 1. Иерархия проекта

## **Алгоритм приведения дроби к несократимому виду**

Алгоритм приведения дроби к нескратимому виду работает на основе алгоритма Эвклида для поиска наибольшего общего делителя.

Алгоритм Эвклида возвращает наибольший общий делитель для числителя и знаменателя рационального числа, затем числитель и знаменатель делятся без остатка на это число. Таким образом, дробь сокращается, и становится несократимой.

Алгоритм Эвклида был реализован в первой лабораторной работе и находится в пространстве имен библиотеки BigInteger.

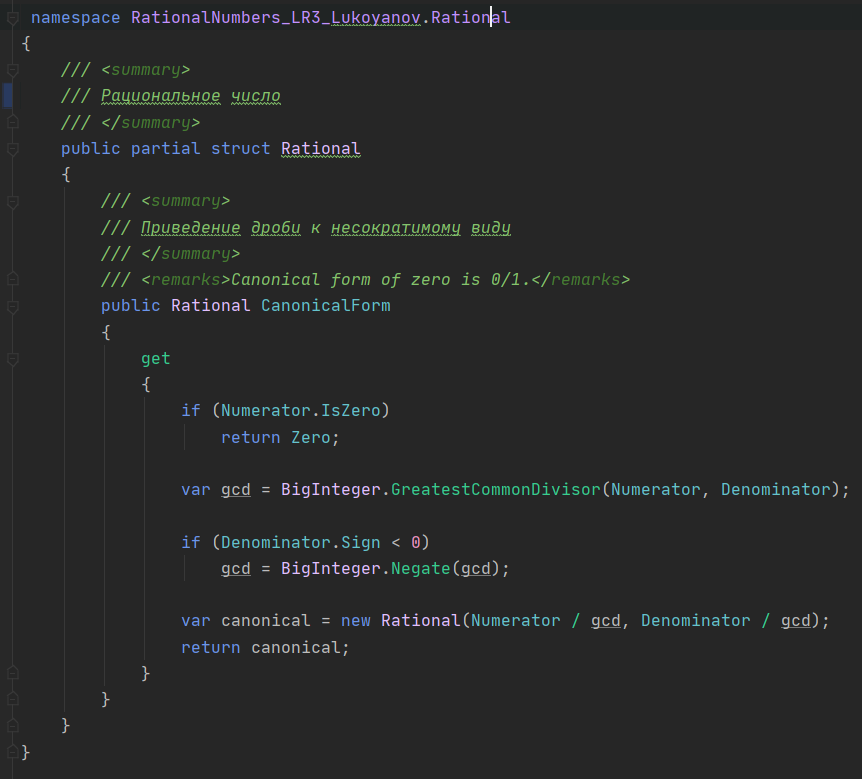


Рисунок 2. Алгоритм приведения дроби к несократимому виду

## **Алгоритм преобразования рациональной дроби в периодическое число с плавающей точкой**

Алгоритм преобразования рациональной дроби в периодическое число с плавающей точкой работает следующим образом: сначала рациональное число преобразовывается в число с плавающей точкой путем деления числителя на знаменатель стандартными средствами языка. Затем происходит перебор всех цифр после запятой и составляются возможные последовательности цифр, которые повторяются. Затем последовательности, между которыми в последовательности есть промежутки, отсеиваются. Оставшиеся последовательности сортируются в порядке удаленности от десятичной точки, и в конце концов, выбирается ближайшая к десятичной точке. Эта последовательность цифр и является периодом числа.

Таким образом, после всех проделанных манипуляций метод возвращает три числа: целую часть числа, часть десятичной дроби после десятичной точки и перед тем, как начнется период, и сам период. Если какая-то часть отсутствует в десятичном представлении числа, на его месте возвращается -1.

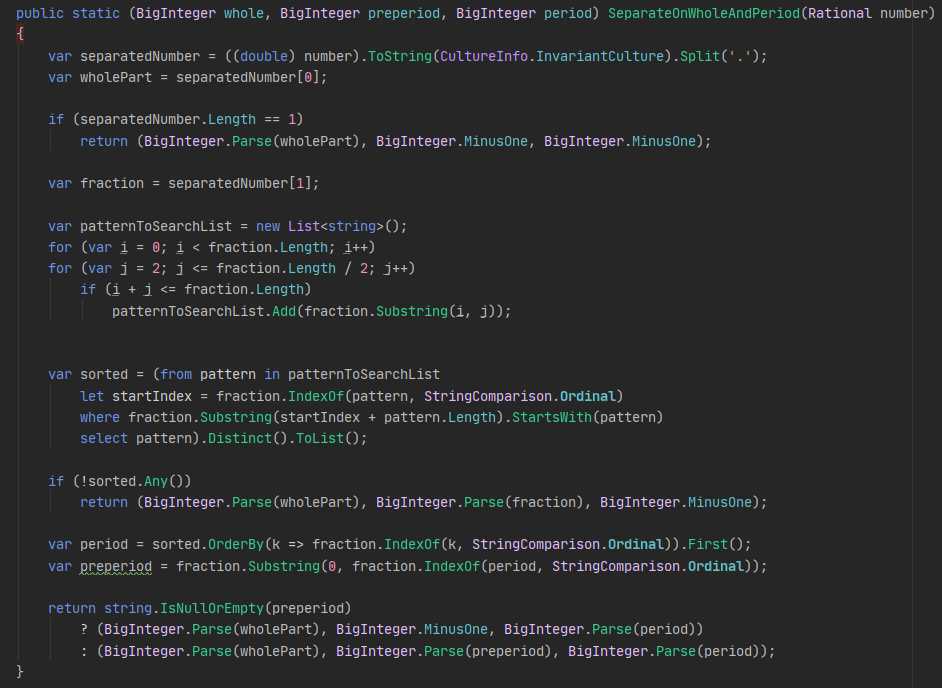


Рисунок 3. Алгоритм разложения числа на целую часть и период

**Алгоритм преобразования периодической дроби в обыкновенную**

Алгоритм преобразования периодической дроби в обыкновенную работает следующим образом.

Число с плавающей точкой, имеющее период умножается на 10 до тех пор, пока дробная часть числа не будет совпадать с началом периода числа. Это число и количество умножений на 10 сохраняется.

Далее мы продолжаем умножать это число на 10, пока опять не получим совпадение дробной части числа с периодом. Снова сохраняем число количество умножений на 10.

Затем c помощью несложной математики мы избавляемся от дробных чисел, и возвращаем рациональную дробь.

Пример работы алгоритма продемонстрирован ниже:

Реализация алгоритма представлена ниже:

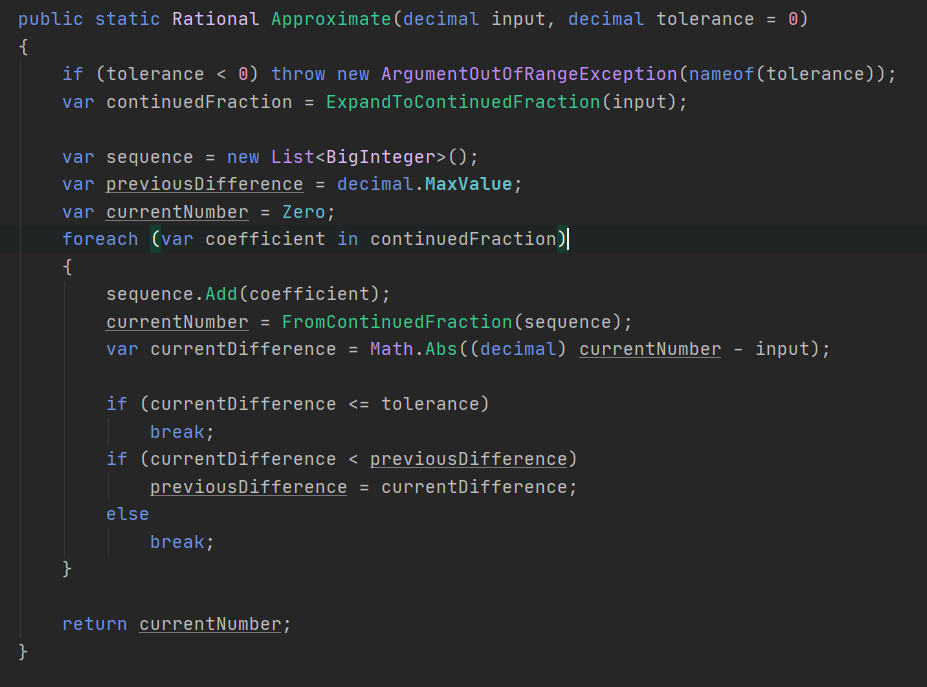


Рисунок 4. Алгоритм преобразования десятичной дроби в рациональное число

# **Инструкция для пользователя**

1. Исходный код размещен в репозитории на GitHub по ссылке: <https://github.com/sharphurt/RationalNumbers_LR3_Lukoyanov>, его необходимо склонировать на свой компьютер.
2. Далее необходимо средствами вашей IDE подключить библиотеку к проекту и запустить сборку всего решения

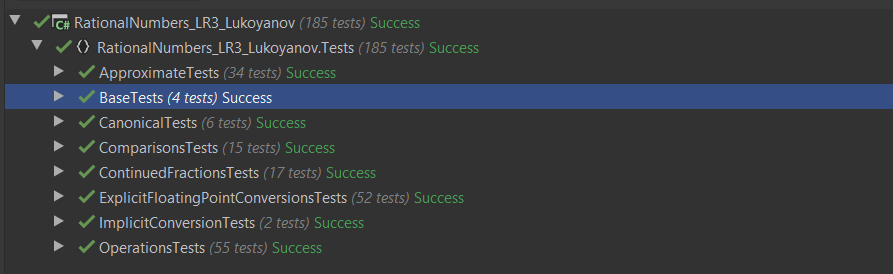
## **Инструкция для программиста**

Исходный код представлен на GitHub по ссылке выше. Вы можете сделать форк репозитория и внести свой вклад в разработку этой библиотеки. Все методы в коде задокументированы и имеют пояснения в виде комментариев.

# **Тестирование**

Для тестов была использована библиотека, предоставляющая функционал для юнит-тестирования NUnit. Тесты охватывают все основные методы для арифметических операций, операций сравнения и служебных методов. Обрабатывают также граничные случаи, такие как: знаменатель, равный 0, попытку деления на 0, случаи переполнения и другие. Всего в проекте 185 тестовых случаев.

Некоторые выдержки из кода тестирования представлены в приложении.



**Вывод**

В ходе работы мною были изучены и реализованы на языке C# алгоритмы работы с рациональными числами, алгоритмы преобразования их в бесконечные периодические дроби и наоборот. Результатом работы стало написание библиотеки, которую можно подключить к любому проекту и использовать в своих целях.

# **Литература**

1. Рациональные числа [Электронный ресурс] Режим доступа: <http://www.cleverstudents.ru/numbers/rational_numbers.html>
2. Rational Numbers & Periodic Decimal Expansions [Электронный ресурс] Режим доступа: <https://homepage.cs.uiowa.edu/~fleck/ratnote.pdf>
3. Compute the period of a decimal number a priori [Электронный ресурс] Режим доступа: <https://math.stackexchange.com/questions/140583/compute-the-period-of-a-decimal-number-a-priori>
4. RationalNumbers\_LR3\_Lukoyanov [Электронный ресурс] Режим доступа: <https://github.com/sharphurt/RationalNumbers_LR3_Lukoyanov>

Приложение

Тестирование

[Fact]  
public void Multiplication3()  
{  
 *// arrange* var left = (Rational.Rational)7 / 4;  
 var right = (Rational.Rational)0 / 3;  
  
 *// action* var product = left \* right;  
  
 *// assert* Assert.Equal(Rational.Rational.Zero, product);  
}  
  
[Fact]  
public void Multiplication4()  
{  
 *// arrange* var left = (Rational.Rational)3 / 4;  
 var right = (Rational.Rational)1 / -3;  
  
 *// action* var product = left \* right;  
  
 *// assert* Assert.Equal(-(Rational.Rational)3 / 12, product);  
}  
  
[Fact]  
public void Negation1()  
{  
 *// arrange* var p = new Rational.Rational(1, 2);  
  
 *// action* var q = -p;  
  
 *// assert* Assert.Equal(-(Rational.Rational)1 / 2, q);  
}  
  
[Fact]  
public void Negation2()  
{  
 *// arrange* var p = new Rational.Rational(-1, 2);  
  
 *// action* var q = -p;  
  
 *// assert* Assert.Equal((Rational.Rational)1 / 2, q);  
}  
  
[Fact]  
public void Negation3()  
{  
 *// arrange* var p = new Rational.Rational(1, -2);  
  
 *// action* var q = -p;  
  
 *// assert* Assert.Equal((Rational.Rational)1 / 2, q);  
}  
  
[Fact]  
public void Negation4()  
{  
 *// arrange* var p = new Rational.Rational(-1, -2);  
  
 *// action* var q = -p;  
  
 *// assert* Assert.Equal(-(Rational.Rational)1 / 2, q);  
}  
  
[Fact]  
public void Negation5()  
{  
 *// arrange* var p = new Rational.Rational(0, 10);  
  
 *// action* var q = -p;  
  
 *// assert* Assert.Equal(p, q);  
}  
  
[Fact]  
public void Subtraction1()  
{  
 *// arrange* var left = (Rational.Rational)3 / 4;  
 var right = (Rational.Rational)1 / 3;  
  
 *// action* var result = left - right;  
  
 *// assert* Assert.Equal((Rational.Rational)5 / 12, result);  
}  
  
[Fact]  
public void Subtraction2()  
{  
 *// arrange* var left = (Rational.Rational)3 / 4;  
 var right = (Rational.Rational)4 / 3;  
  
 *// action* var result = left - right;  
  
 *// assert* Assert.Equal(-(Rational.Rational)7 / 12, result);  
}  
  
[Fact]  
public void Subtraction3()  
{  
 *// arrange* var left = (Rational.Rational)0 / 4;  
 var right = (Rational.Rational)4 / 3;  
  
 *// action* var result = left - right;  
  
 *// assert* Assert.Equal(-(Rational.Rational)4 / 3, result);  
}  
  
[Fact]  
public void Subtraction4()  
{  
 *// arrange* var left = (Rational.Rational)5 / 4;  
 var right = (Rational.Rational)0 / 3;  
  
 *// action* var result = left - right;  
  
 *// assert* Assert.Equal((Rational.Rational)5 / 4, result);  
}  
  
[Fact]  
public void Subtraction5()  
{  
 *// arrange* var p = (Rational.Rational)5 / 4;  
  
 *// action* var result = p - p;  
  
 *// assert* Assert.Equal(Rational.Rational.Zero, result);  
}  
  
[Fact]  
public void Absolute1()  
{  
 *// arrange* var p = new Rational.Rational(1, 2);  
  
 *// action* var q = -p;  
  
 *// assert* Assert.Equal((Rational.Rational)1 / 2, Rational.Rational.Abs(q));  
}

public class ApproximateTests  
{  
 [Theory]  
 [InlineData(0.333f, 0.0001f, 333, 1000)]  
 [InlineData(-0.333f, 0.0001f, -333, 1000)]  
 [InlineData(0.333f, 0.001f, 1, 3)]  
 [InlineData(-0.333f, 0.001f, -1, 3)]  
 [InlineData(2.5f, 0f, 5, 2)]  
 [InlineData(-2.5f, 0f, -5, 2)]  
 [InlineData(-1f, 0f, -1, 1)]  
 [InlineData(1f, 0f, 1, 1)]  
 public void ApproximateFloat(float input, float tolerance, int expectedNumerator, int expectedDenominator)  
 {  
 *// action* var rational = Rational.Rational.Approximate(input, tolerance);  
  
 *// assert* Assert.Equal((Rational.Rational)expectedNumerator / expectedDenominator, rational);  
 }  
  
 [Theory]  
 [InlineData(0.333d, 0.0001d, 333, 1000)]  
 [InlineData(-0.333d, 0.0001d, -333, 1000)]  
 [InlineData(0.333d, 0.001d, 1, 3)]  
 [InlineData(-0.333d, 0.001d, -1, 3)]  
 [InlineData(2.5d, 0d, 5, 2)]  
 [InlineData(-2.5d, 0d, -5, 2)]  
 [InlineData(-1d, 0d, -1, 1)]  
 [InlineData(1d, 0d, 1, 1)]  
 public void ApproximateDouble(double input, double tolerance, int expectedNumerator, int expectedDenominator)  
 {  
 *// action* var rational = Rational.Rational.Approximate(input, tolerance);  
  
 *// assert* Assert.Equal((Rational.Rational)expectedNumerator / expectedDenominator, rational);  
 }  
  
 [Theory]  
 [InlineData(3, 1, 1.42E-1)]  
 [InlineData(22, 7, 1.27E-3)]  
 [InlineData(333, 106, 8.33E-5)]  
 [InlineData(355, 113, 2.67E-7)]  
 [InlineData(103993, 33102, 5.78E-10)]  
 [InlineData(104348, 33215, 3.32E-10)]  
 [InlineData(208341, 66317, 1.23E-10)]  
 [InlineData(312689, 99532, 2.92E-11)]  
 [InlineData(833719, 265381, 8.72E-12)]  
 [InlineData(1146408, 364913, 1.62E-12)]  
 public void ApproximatePI(int expectedNumerator, int expectedDenominator, double tolerance)  
 {  
 *// action* var rational = Rational.Rational.Approximate(Math.**PI**, tolerance);  
  
 *// assert* Assert.Equal((Rational.Rational)expectedNumerator / expectedDenominator, rational);  
 }  
  
 [Theory]  
 [InlineData("0.333", "0.0001", 333, 1000)]  
 [InlineData("-0.333", "0.0001", -333, 1000)]  
 [InlineData("0.333", "0.001", 1, 3)]  
 [InlineData("-0.333", "0.001", -1, 3)]  
 [InlineData("2.5", "0", 5, 2)]  
 [InlineData("-2.5", "0", -5, 2)]  
 [InlineData("-1", "0", -1, 1)]  
 [InlineData("1", "0", 1, 1)]  
 public void ApproximateDecimal(string inputStr, string toleranceStr, int expectedNumerator, int expectedDenominator)  
 {  
 *// arrange* var input = decimal.Parse(inputStr, CultureInfo.InvariantCulture);  
 var tolerance = decimal.Parse(toleranceStr, CultureInfo.InvariantCulture);  
  
 *// action* var rational = Rational.Rational.Approximate(input, tolerance);  
  
 *// assert* Assert.Equal((Rational.Rational)expectedNumerator / expectedDenominator, rational);  
 }  
}