Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
Высшая школа компьютерных технологий и информационных систем

Реферат

Дисциплина: Компьютерная Алгебра

Тема: «Разработка библиотеки функций для обработки символьных представлений математических объектов, имеющих алгебраическую структуру - кольцо»

Выполнили студенты гр. 5130901/20201
К.М. Зезина ______________________
Н.А. Дюков ______________________
студенты гр. 5130901/20101
А. Теллили ______________________
А.Д. Усачев ______________________
студенты гр. 5130901/20102
О.С. Соловьев ______________________
А.А. Вагнер ______________________
студенты гр. 5130901/20103
С.А. Мукий ______________________
Ф.Г. Кудрин ______________________

(подпись)

Преподаватель __________________________ И.А. Малышев
(подпись)

“_____” _____________ 2024 г.

Санкт-Петербург

2024

Оглавление

Понятие алгебраической структуры «кольцо»

Для того, чтобы изучать кольца, необходимо рассмотреть структуру **абелевой группы**.

Сложение вещественных чисел обладает следующими свойствами:

$a + b = b + a$ (коммутативность)
$(a + b) + c = a + (b + c)$ (ассоциативность)
$a + 0 = a$
$a + (-a) = 0$

Умножение вещественных чисел обладает аналогичными свойствами:
$ab = ba$ (коммутативность)
$(ab)c = a(bc)$ (ассоциативность)
$a1 = a$
$aa^{-1} = 1$ при $a \neq 0$

Кольца — это алгебраические структуры с двумя операциями: сложением и умножением. Их аксиомы, как и аксиомы абелевой группы, подсказаны свойствами операций над вещественными числами. При этом аксиомы кольца — это разумный минимум требований относительно свойств операций, позволяющий охватить и другие важные примеры алгебраических структур, например, множество векторов пространства с операциями сложения и векторного умножения.

Кольцом называется множество К с операциями сложения и умножения, обладающим следующими свойствами:

- Относительно сложения К есть абелева группа (называемая аддитивной группой кольца К)
- $a(b + c) = ab + ac$ и $(a + b)c = ac + bc$ для любых $a, b, c \in K$ (дистрибутивность умножения относительно сложения)

Выведем некоторые следствия аксиом кольца, не входящие в число следствий аксиом аддитивной абелевой группы.

- $a0 = 0a = 0$ для любого $a \in K$. Пусть $a0 = b$, тогда
$$b + b = a0 + a0 = a(0 + 0) = a0 = b$$
Откуда: $b = b - b = 0$
Аналогично доказывается, что $0a = 0$
- $a(-b) = (-a)b = -ab$ для любых $a, b \in K$. Отсюда:
$$ab + a(-b) = a\big(b + (-b)\big) = a0 = 0$$
и, аналогично, $ab + a(-b) = 0$

- $a(b-c) = ab - ac$ и $(a-b)c = ac - bc$ для любых $a, b, c \in K$
  Отсюда: $a(b-c) + ac = a(b-c+c) = ab$
  и, аналогично, $(a-b)c + bc = ac$

Кольцо К называется *коммутативным*, если умножение в нем коммутативно:

$$ab = ba \qquad \forall a, b$$

и *ассоциативным*, если умножение в нем ассоциативно:

$$(ab)c = a(bc) \qquad \forall a, b, c$$

Элемент 1 кольца называется *единицей*, если: $a1 = 1a = a \quad \forall a$

Так же, как в случае мультипликативной абелевой группы, доказывается, что в кольце не может быть двух различных единиц (но может не быть ни одной).

Замечание 1.

Если 1 = 0, то для любого а имеем: $a = a1 = a0 = 0$, то есть кольцо состоит из одного нуля. Если кольца содержит более одного элемента, то $1 \neq 0$.

Замечание 2.

При наличии коммутативности из двух тождеств дистрибутивности, входящих в определение кольца, можно оставить лишь одно. Аналогичное замечание относится к определению единицы.

- Числовые множества $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$ являются коммутативными ассоциативными кольцами с единицей относительно обычных операций сложения и умножения.
- Множество $2\mathbb{Z}$ четных чисел является коммутативным ассоциативным кольцом без единицы.
- Множество всех функций, определенных на заданном подмножестве числовой прямой, является коммутативным ассоциативным кольцом с единицей относительно обычных операций сложения и умножения функций.
- Множество векторов пространства с операциями сложения и векторного умножения является некоммутативным и неассоциативным кольцом. Однако в нем выполняются следующие тождества, которые в некотором смысле заменяют коммутативность и ассоциативность:
  $$ab + ba = 0 \text{ (антикоммутативность)}$$
  $$(ab)c + (bc)a + (ca)b = 0 \text{ (Тождество Якоби)}$$

Описание библиотеки функций для кольца

Поставленное задание:

Разработать программную библиотеку функций для обработки символьных представлений математических объектов, имеющих заданную алгебраическую структуру (Вариант 8. Кольцо)

Библиотека должна обеспечивать:

1. Создание и удаление объекта
2. Редукция (упрощение) выражений, содержащих объекты и их свойства
3. Копирование объекта
4. Построение выражений, содержащих объектов и их свойства
5. Определение и переопределение свойств объекта

Часть 1. Создание и удаление объекта

```
# Создание
def __init__(self, coefficients):
    self.coefficients = coefficients
    self.reduce()

# Удаление
def delete(self):
    self.coefficients = None
```

Листинг 1. Создание и удаление объектов

Для реализации алгебраической структуры был написан класс Ring (с полным листингом кода можно ознакомиться в Приложении 1).
Начальная и одна из самых главных функций – функция создания объекта класса `def __init`.
В функцию подаются значения одного полинома: в порядке индекса – это степень при их

Пример:

```
p1 = Ring([1, 2])  # 2x + 1
p2 = Ring([3, 1])  # x + 3
p3 = Ring([0, 1])  # x
```

В самой функции присваиваем объекту класса значения коэффициентов, а после вызываем функцию **reduce** для объекта класса.

Часть 2. Редукция (упрощение) выражений, содержащих объекты и их свойства

```
# Убирает лишние нули если есть
def reduce(self):
    while self.coefficients and self.coefficients[-1] == 0:
        self.coefficients.pop()
    return self
```

Листинг 2. Упрощение выражений

В ходе создания объекта мы автоматически обращаемся к функции **reduce**. Комментарий в коде гласит: `# Убирает лишние нули если есть`. Что это значит? В случае, если на вход нам подается следующая строка:

```
p4 = Ring([0, 0, 0, 3, 0, 0]) # 3x^3 + 0x^2 + 0x + 0
```

То при выводе мы получим: `3x^3`

Это все говорит о том, что числа справа от 3 (то есть $0x^4 + 0x^5$) не являются значимыми и мы их удаляем из-за отсутствия необходимости. Однако, может показаться на первый взгляд, что мы не учитываем и числа слева $(0 + 0x + 0x^2)$, но это не так, ибо они все еще хранятся в данных объекта класса и не выводятся на экран, чтобы не заполнять пространство, но все еще учитываются при дальнейших вычислениях.

Далее рассмотрим функцию **delete**. Она полностью удаляет объект класса, зачищая коэффициенты полинома, определяя их как `None.`

Часть 3. Копирование объекта

```
from copy import deepcopy

#Копирование текущего объекта кольца.
# return: Новый объект кольца, идентичный текущему.

def copy(self):
    return deepcopy(self)
```

Листинг 3. Копирование объекта

Копирование объекта класса необходимо в том случае, если нужно использовать тот же самый полином, который был записан ранее для дальнейших манипуляций.

Функция deepcopy работает следующими путями:

- Хранения "memo" словаря объектов, скопированных во время текущего прохода копирования;
- Позволения классам, определенным пользователем, переопределять операцию копирования или набор копируемых компонентов.

Пример работы функции:

```
r = [1, 2, 3]
r.append(r)
print(r) # [1, 2, 3, [1, 2, 3]]
p = copy.deepcopy(r)
print(p) # [1, 2, 3, [1, 2, 3]]
```

Часть 4. Построение выражений, содержащих объектов и их свойства

```
# Сложение
def __add__(self, other):
    max_len = max(len(self.coefficients),
len(other.coefficients))
    padded_self = self.coefficients + [0] * (max_len -
len(self.coefficients))
    padded_other = other.coefficients + [0] * (max_len -
len(other.coefficients))
    result_coeffs = [a + b for a, b in zip(padded_self,
padded_other)]
    return Ring(result_coeffs)

# Умножение
def __mul__(self, other):
    result_coeffs = [0] * (len(self.coefficients) +
len(other.coefficients) - 1)
    for i, a in enumerate(self.coefficients):
        for j, b in enumerate(other.coefficients):
            result_coeffs[i + j] += a * b
    return Ring(result_coeffs)

# Сравнение
def __eq__(self, other):
    # Приводим коэффициенты к одинаковой длине
    max_len = max(len(self.coefficients),
len(other.coefficients))
    padded_self = self.coefficients + [0] * (max_len -
len(self.coefficients))
    padded_other = other.coefficients + [0] * (max_len -
len(other.coefficients))
    return padded_self == padded_other
```

Листинг 4. Свойства кольца

Как мы помним, кольца — это алгебраические структуры с двумя операциями: сложением и умножением. Поэтому опишем эти функции, согласно структуре кольца, описанной ранее на странице 3, путем работы с коэффициентами.

Сложение:

__add__ позволяет осуществлять сложение двух объектов. Это специальный метод, который вызывается при использовании оператора +.

max_len вычисляется как максимальная длина коэффициентов двух объектов. Это нужно для обеспечения правильного выравнивания при сложении.

padded_self и padded_other создаются, чтобы уравнять длины двух списков коэффициентов (если один из них короче, добавляются нули в конец).

result_coeffs создается с использованием спискового включения. Для каждой пары коэффициентов (из обоих списков, выровненных по длине) производится суммирование соответствующих элементов.

Умножение:

__mul__ позволяет осуществлять умножение двух объектов. Это специальный метод, который вызывается при использовании оператора *

result_coeffs инициализируется списком нулей, длина которого равна сумме длин коэффициентов обоих объектов минус 1. Это соответствует правилу о том, что произведение двух многочленов степени n и m будет многочленом степени n + m.

Используются два вложенных цикла: внешний цикл перебирает коэффициенты первого объекта, а внутренний – второго.
Для каждой пары коэффициентов a и b, умножение происходит и добавляется к соответствующему индексу в result_coeffs (индекс равен сумме индексов i и j).

Сравнение:

__eq__ позволяет использовать оператор == для сравнения двух объектов этого класса. Это специальный метод, вызываемый при выполнении операции сравнения.

max_len вычисляется как максимальная длина коэффициентов двух объектов. Это необходимо для корректного выравнивания списков перед сравнением.

padded_self создается путём добавления нулей в конец списка коэффициентов первого объекта до достижения максимальной длины.
padded_other аналогичным образом создается для второго объекта.

Метод возвращает результат сравнения двух списков коэффициентов: padded_self и padded_other. Если оба списка равны, возвращается True, в противном случае — False.

Проведем проверку свойств:

```python
def test_polynomial_ring():
    # Создаем полиномы
    p1 = Ring([1, 2])  # 2x + 1
    p2 = Ring([3, 1])  # x + 3
    p3 = Ring([0, 1])  # x
    p0 = Ring([0])     # 0
    one = Ring([1])  # 1

    print("1. Проверка коммутативности сложения")
    print(f"p1 + p2 = p2 + p1  => {p1+p2} = {p2+p1}")
    if (p1 + p2) != (p2 + p1):
        print("Ошибка: Сложение не коммутативно\n")
    else:
        print("Свойство коммутативности сложения не нарушено\n")

    print("2. Проверка ассоциативности сложения")
    print(f"(p1 + p2) + p3 = p1 + (p2 + p3)  => {(p1 + p2) + p3} = {p1 + (p2
+ p3)}")
    if (p1 + p2) + p3 != p1 + (p2 + p3):
        print("Ошибка: Сложение не ассоциативно\n")
    else:
        print("Свойство ассоциативности сложения не нарушено\n")

    print("3. Проверка существования нуля")
    print(f"p1 * p0 = p0  => {p1 * p0} = {p0}")
    if p1 * p0 != p0:
        print("Ошибка: Не существует нуля\n")
    else:
        print("Свойство существования нуля не нарушено\n")

    print("4. Проверка коммутативности умножения")
    print(f"p1 * p2 = p2 * p1  => {p1 * p2} = {p2 * p1}")
    if (p1 * p2) != (p2 * p1):
        print("Ошибка: Умножение не коммутативно\n")
    else:
        print("Свойство коммутативности умножения не нарушено\n")

    #
    print("5. Проверка ассоциативности умножения")
    print(f"(p1 * p2) * p3 = p1 * (p2 * p3)  => {(p1 * p2) * p3} = {p1 * (p2
* p3)}")
    if (p1 * p2) * p3 != p1 * (p2 * p3):
        print("Ошибка: Умножение не ассоциативно\n")
    else:
        print("Свойство ассоциативности умножения не нарушено\n")

    print("6. Проверка дистрибутивности")
    print(f"p1 * (p2 + p3) = (p1 * p2) + (p1 * p3)  => {p1 * (p2 + p3)} =
{(p1 * p2) + (p1 * p3)}")
    if p1 * (p2 + p3) != (p1 * p2) + (p1 * p3):
        print("Ошибка: Умножение не дистрибутивно\n")
    else:
        print("Свойство дистрибутивности не нарушено\n")

    print("7.  Проверка существования единицы")
    print(f" p1 * one = p1  => {p1 * one} = {p1}")
    if p1 * one != p1:
        print("Ошибка: Не существует единицы\n")
    else:
          print("Свойство существования единицы не нарушено\n")
```

Листинг 5. Проверка свойств объекта класса

```
1. Проверка коммутативности сложения
p1 + p2 = p2 + p1  => 4 + 3x = 4 + 3x
Свойство коммутативности сложения не нарушено

2. Проверка ассоциативности сложения
(p1 + p2) + p3 = p1 + (p2 + p3)  => 4 + 4x = 4 + 4x
Свойство ассоциативности сложения не нарушено

3. Проверка существования нуля
p1 * p0 = p0  => 0 = 0
Свойство существования нуля не нарушено

4. Проверка коммутативности умножения
p1 * p2 = p2 * p1  => 3 + 7x + 2x^2 = 3 + 7x + 2x^2
Свойство коммутативности умножения не нарушено
```

```
5. Проверка ассоциативности умножения
(p1 * p2) * p3 = p1 * (p2 * p3)  => 3x + 7x^2 + 2x^3 = 3x + 7x^2 + 2x^3
Свойство ассоциативности умножения не нарушено

6. Проверка дистрибутивности
p1 * (p2 + p3) = (p1 * p2) + (p1 * p3)  => 3 + 8x + 4x^2 = 3 + 8x + 4x^2
Свойство дистрибутивности не нарушено

7.  Проверка существования единицы
 p1 * one = p1  => 1 + 2x = 1 + 2x
Свойство существования единицы не нарушено
```

Рис. 1. Результаты тестирования свойств объекта

Как мы можем видеть, все выражения со свойствами объекта типа кольца определены корректно и работают исправно.

Часть 5. Определение и переопределение свойств объекта

Так как свойства кольца – это умножение и сложение, переопределить их не удастся. Поэтому определим в классе Ring функцию преобразования выражения в строку для оптимальности вывода полинома, вместо вывода его в качестве массива значений в порядке определения индексной степени коэффициентов:

```python
# Преобразование в строку
def __str__(self):
    terms = []
    for i, coeff in enumerate(self.coefficients):
        if coeff != 0:
            if coeff == 1 and i > 0:
                coeff_str = ""
            elif coeff == -1 and i > 0:
                coeff_str = "-"
            else:
```

```
                coeff_str = str(coeff)
            if i == 0:
                terms.append(coeff_str)
            elif i == 1:
                terms.append(f"{coeff_str}x")
            else:
                terms.append(f"{coeff_str}x^{i}")
    if not terms:
        return "0"
    return " + ".join(terms)
```

Листинг 5. Приведение полинома к строчному виду

Вывод:

В ходе проделанной работы удалось ознакомиться с алгебраической структурой – кольцом. После была написана библиотека, которая содержит в себе все свойства структуры, а также были добавлены функции создания и удаления объекта, его редукция (упрощение), копирования объекта, построения выражений, содержащих объектов и их свойства, а также определения и переопределения свойств объекта. Все участники команды принимали активное участие в разработке проекта, поэтому этот опыт был полезен не только с точки зрения обучающего процесса в рамках дисциплины «Компьютерная алгебра», но и с точки зрения умения работать в сплоченной команде.

Приложение 1. Листинг класса Ring

```
from copy import deepcopy


class Ring:
    # Создание
    def __init__(self, coefficients):
        self.coefficients = coefficients
        self.reduce()
    # Сложение
    def __add__(self, other):
        max_len = max(len(self.coefficients),
len(other.coefficients))
        padded_self = self.coefficients + [0] * (max_len -
len(self.coefficients))
        padded_other = other.coefficients + [0] * (max_len -
len(other.coefficients))
        result_coeffs = [a + b for a, b in zip(padded_self,
padded_other)]
        return Ring(result_coeffs)

    # Умножение
    def __mul__(self, other):
        result_coeffs = [0] * (len(self.coefficients) +
len(other.coefficients) - 1)
        for i, a in enumerate(self.coefficients):
            for j, b in enumerate(other.coefficients):
                result_coeffs[i + j] += a * b
        return Ring(result_coeffs)

    # Преобразование в строку
    def __str__(self):
        terms = []
        for i, coeff in enumerate(self.coefficients):
            if coeff != 0:
                if coeff == 1 and i > 0:
                    coeff_str = ""
                elif coeff == -1 and i > 0:
                    coeff_str = "-"
                else:
                    coeff_str = str(coeff)
                if i == 0:
                    terms.append(coeff_str)
                elif i == 1:
                    terms.append(f"{coeff_str}x")
                else:
                    terms.append(f"{coeff_str}x^{i}")
        if not terms:
            return "0"
        return " + ".join(terms)

    # Сравнение
```

```
    def __eq__(self, other):
        # Приводим коэффициенты к одинаковой длине
        max_len = max(len(self.coefficients),
len(other.coefficients))
        padded_self = self.coefficients + [0] * (max_len -
len(self.coefficients))
        padded_other = other.coefficients + [0] * (max_len -
len(other.coefficients))
        return padded_self == padded_other

    # Убирает лишние нули если есть
    def reduce(self):
        while self.coefficients and self.coefficients[-1] == 0:
            self.coefficients.pop()
        return self

    #Копирование текущего объекта кольца.
    # return: Новый объект кольца, идентичный текущему.

    def copy(self):
        return deepcopy(self)

    # Удаление
    def delete(self):
        self.coefficients = None
```

Приложение 2. Листинг тестирующего program-file

```
from ring import Ring


def test_polynomial_ring():
    # Создаем полиномы
    p1 = Ring([1, 2])  # 2x + 1
    p2 = Ring([3, 1])  # x + 3
    p3 = Ring([0, 1])  # x
    #p4 = Ring([0, 0, 0, 3, 0, 0]) # 3x^3 + 0x^2 + 0x + 0
    p0 = Ring([0])     # 0
    one = Ring([1])  # 1

    print("1. Проверка коммутативности сложения")
    print(f"p1 + p2 = p2 + p1  => {p1+p2} = {p2+p1}")
    if (p1 + p2) != (p2 + p1):
        print("Ошибка: Сложение не коммутативно\n")
    else:
        print("Свойство коммутативности сложения не нарушено\n")

    print("2. Проверка ассоциативности сложения")
    print(f"(p1 + p2) + p3 = p1 + (p2 + p3)  => {(p1 + p2) + p3}
= {p1 + (p2 + p3)}")
    if (p1 + p2) + p3 != p1 + (p2 + p3):
        print("Ошибка: Сложение не ассоциативно\n")
    else:
        print("Свойство ассоциативности сложения не нарушено\n")

    print("3. Проверка существования нуля")
    print(f"p1 * p0 = p0  => {p1 * p0} = {p0}")
    if p1 * p0 != p0:
        print("Ошибка: Не существует нуля\n")
    else:
        print("Свойство существования нуля не нарушено\n")

    print("4. Проверка коммутативности умножения")
    print(f"p1 * p2 = p2 * p1  => {p1 * p2} = {p2 * p1}")
    if (p1 * p2) != (p2 * p1):
        print("Ошибка: Умножение не коммутативно\n")
    else:
        print("Свойство коммутативности умножения не
нарушено\n")

    #
    print("5. Проверка ассоциативности умножения")
    print(f"(p1 * p2) * p3 = p1 * (p2 * p3)  => {(p1 * p2) * p3}
= {p1 * (p2 * p3)}")
    if (p1 * p2) * p3 != p1 * (p2 * p3):
        print("Ошибка: Умножение не ассоциативно\n")
    else:
        print("Свойство ассоциативности умножения не
нарушено\n")
```

```
    print("6. Проверка дистрибутивности")
    print(f"p1 * (p2 + p3) = (p1 * p2) + (p1 * p3)  => {p1 * (p2
+ p3)} = {(p1 * p2) + (p1 * p3)}")
    if p1 * (p2 + p3) != (p1 * p2) + (p1 * p3):
        print("Ошибка: Умножение не дистрибутивно\n")
    else:
        print("Свойство дистрибутивности не нарушено\n")

    print("7.  Проверка существования единицы")
    print(f" p1 * one = p1  => {p1 * one} = {p1}")
    if p1 * one != p1:
        print("Ошибка: Не существует единицы\n")
    else:
        print("Свойство существования единицы не нарушено\n")


# Запуск тестов
test_polynomial_ring()
```