|  |  |  |
| --- | --- | --- |
| МИНОБРНАУКИ РОССИИ | | |
| Федеральное государственное автономное  образовательное учреждение высшего образования  «Пермский государственный национальный  исследовательский университет» | | |
|  | Институт компьютерных наук и технологий | |
| **ОТЧЁТ**  по индивидуальной работе №2  по дисциплине «Язык программирования C++»  Вариант 5 | | |
|  | | Работу выполнил  студент группы ИТ-6-2024 1 курса  Трянин Даниил Сергеевич Фамилия.И.О.  «\_\_\_» \_\_\_\_\_\_\_\_2025 г. |
| Работу проверил  \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Фамилия И.О.  «\_\_\_» \_\_\_\_\_\_\_\_\_ 2025 г. |
| Пермь 2025 | | |

СОДЕРЖАНИЕ

[Постановка задачи 3](#_Toc201184390)

[Алгоритм решения 3](#_Toc201184391)

[Тестирование 3](#_Toc201184392)

[Код программы 3](#_Toc201184393)

# Постановка задачи

Разработать программу на языке Python, которая находит в строке цифр, рассматриваемой как кольцо, такие три последовательных числа A, B и C, что A + B = C. Цифры должны использоваться в порядке следования в кольце и только один раз. Решение должно соответствовать принципам ООП, использовать пользовательские структуры данных и обработку исключений. Запрещено использование стандартных библиотек структур данных.

# Алгоритм решения

Для решения задачи используется структура данных «кольцевой односвязный список», реализованная вручную через классы CircularListNode и CircularLinkedList. Эта структура выбрана потому, что строка цифр рассматривается как кольцо: после последнего символа снова идёт первый, и нам необходимо многократно проходить по кругу с произвольной позиции. В отличие от обычного списка, кольцевой список позволяет удобно реализовать такую циклическую навигацию. На первом этапе программа строит кольцевой список из входной строки. Каждый узел содержит одну цифру и ссылку на следующий элемент, а последний узел ссылается на первый, замыкая кольцо. Это позволяет обходить структуру с любой позиции и формировать последовательности чисел по кругу. Затем начинается основной перебор всех возможных разбиений кольца на три числа — слагаемое A, слагаемое B и сумму C. Для этого выбираются все допустимые комбинации длин чисел A, B и C. Длины выбираются в разумных пределах, например от 1 до 30 символов, чтобы избежать избыточного перебора. Условие состоит в том, что суммарная длина A, B и C не должна превышать длину кольца. Для каждой тройки длин происходит полный обход всех возможных стартовых позиций в кольце. Это обеспечивает проверку всех вариантов, где A начинается с любого узла списка. С каждой позиции извлекаются три числа подряд: сначала число A заданной длины, затем сразу после него число B, и затем число C. Извлечение происходит последовательно по узлам списка, с соблюдением порядка и длины. Полученные строки цифр преобразуются в целые числа. После этого происходит проверка условия: сумма A и B должна быть равна C. Если это условие выполняется, программа возвращает найденное тождество в виде строки, например 190+20=210, и завершает выполнение. Если ни одна комбинация не даёт верного тождества, программа возвращает строку No.

Особое внимание уделено тому, чтобы каждый узел использовался только один раз в пределах одного кандидата на тождество: A, B и C состоят из непересекающихся и последовательных участков кольца. Также проверяется, чтобы числа не начинались с нуля, если они многозначные. Ключевую роль в алгоритме играет метод get\_numbers\_starting\_from, который из заданной стартовой позиции извлекает три последовательных числа нужной длины, корректно переходя через границу кольца. Для повышения читаемости и модульности этот функционал выделен в отдельный метод класса CircularLinkedList. Преимуществом данной реализации является точное соблюдение условий задачи, в том числе использование исключительно пользовательских структур данных, строгое следование принципам ООП и наличие обработки ошибок. В частности, при создании кольцевого списка реализована проверка на пустую строку и на наличие недопустимых символов. При ошибке программа сообщает об этом пользователю через систему исключений и дружественный интерфейс.

Таким образом, алгоритм гарантирует полный перебор всех допустимых вариантов и корректную проверку условия A + B = C, при этом соблюдая требование работы с кольцом и ограничения на повторное использование цифр.

# Тестирование

Программа протестирована на нескольких примерах:  
-Ввод: 01902021 → Результат: 190+20=210  
-Ввод: 111111 → Результат: No  
-Ввод: 123456789 → Результат: 123+456=579  
-Ввод: 0987654321 → Результат: No  
Все тесты показали корректную работу программы на допустимых данных.

# Код программы

class CircularListNode:

    """Узел кольцевого списка"""

    def \_\_init\_\_(self, value, next\_node=None):

        self.value = value

        self.next = next\_node if next\_node is not None else self

    def \_\_repr\_\_(self):

        return f"Node({self.value})"

class CircularLinkedList:

    """Кольцевой список"""

    def \_\_init\_\_(self, data\_string):

        if not data\_string:

            raise ValueError("Input string cannot be empty")

        if not data\_string.isdigit():

            raise ValueError("Input string must contain only digits")

        self.head = None

        self.length = 0

        self.\_build\_list(data\_string)

    def \_build\_list(self, data\_string):

        """Строит кольцевой список из строки цифр"""

        prev\_node = None

        first\_node = None

        for char in data\_string:

            node = CircularListNode(int(char))

            self.length += 1

            if prev\_node is not None:

                prev\_node.next = node

            else:

                first\_node = node

            prev\_node = node

        if prev\_node is not None:

            prev\_node.next = first\_node

            self.head = first\_node

    def get\_numbers\_starting\_from(self, start\_node, lengths):

        """

        Получает числа из кольца, начиная с start\_node, с заданными длинами

        Возвращает кортеж (число1, число2, число3, следующий узел)

        """

        numbers = []

        current\_node = start\_node

        for length in lengths:

            if length == 0:

                numbers.append(0)

                continue

            num\_str = ""

            for \_ in range(length):

                num\_str += str(current\_node.value)

                current\_node = current\_node.next

            numbers.append(int(num\_str))

        return (\*numbers, current\_node)

    def find\_equation(self):

        """Ищет уравнение вида A+B=C в кольце"""

        max\_length = min(30, self.length // 3)  # Ограничиваем максимальную длину числа

        for a\_len in range(1, max\_length + 1):

            for b\_len in range(1, max\_length + 1):

                c\_len\_max = min(a\_len, b\_len) + 1

                for c\_len in range(max(1, max(a\_len, b\_len) - 1), c\_len\_max + 1):

                    if a\_len + b\_len + c\_len > self.length:

                        continue

                    current\_node = self.head

                    for \_ in range(self.length):

                        a, b, c, next\_node = self.get\_numbers\_starting\_from(

                            current\_node, [a\_len, b\_len, c\_len]

                        )

                        if a + b == c:

                            return f"{a}+{b}={c}"

                        current\_node = current\_node.next

        return "No"

def main():

    """Основная функция программы"""

    print("Программа для поиска уравнения A+B=C в числовом кольце")

    print("Введите строку цифр (без пробелов):")

    try:

        input\_data = input().strip()

        if not input\_data:

            raise ValueError("Введена пустая строка")

        ring = CircularLinkedList(input\_data)

        result = ring.find\_equation()

        print("\nРезультат:")

        print(result)

    except ValueError as e:

        print(f"\nОшибка ввода: {e}")

    except Exception as e:

        print(f"\nПроизошла непредвиденная ошибка: {e}")

if \_\_name\_\_ == "\_\_main\_\_":

    main()