|  |  |  |
| --- | --- | --- |
| 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 Python»  Вариант 3 | | |
|  | | Работу выполнила  студент группы ИТ-7-2025 1 курса  Андреева Л. В.  «10» июня 2024 г. |
| Работу проверил  \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Фамилия И.О.  «\_\_\_» \_\_\_\_\_\_\_\_\_ 2024 г. |
| Пермь 2024 | | |

СОДЕРЖАНИЕ

[Постановка задачи 3](#_Toc153130027)

[Алгоритм решения 4](#_Toc153130028)

[Тестирование 6](#_Toc153130029)

[Код программы 7](#_Toc153130030)

# Постановка задачи

Напечатать в порядке возрастания первых n натуральных чисел, в разложение  
которых на простые множители входят только числа 2,3,5.  
Идея решения: введем три очереди x2, x3, x5 в которых будем хранить  
элементы, которые соответственно в 2, 3, 5 раз больше напечатанных, но еще  
не напечатаны. Рассмотрим наименьший из ненапечатанных элементов: пусть  
это x. Тогда он делится нацело на одно из чисел 2, 3, 5; x находится в одной из  
очередей и является в ней первым элементом (меньшие его уже напечатаны,  
а элементы очередей не напечатаны). Напечатав x, нужно изъять его из  
очереди и добавить в очередь кратные ему элементы. Длины очередей не  
превосходят числа напечатанных элементов. Изначально в очередях хранится  
по одному числу.

# Алгоритм решения

**1. Инициализация очередей:**

* Создание трех очередей.
* Помещение начальных значений в очереди.
* Создание списка для хранения отсортированных чисел и добавление в него начального значения 1.
* Инициализация счетчика для отслеживания количества найденных чисел.

**2. Основной цикл (генерация чисел до достижения n):**

* Повторять до тех пор, пока количество чисел меньше значения n.
* Определить минимальное значение среди первых элементов очередей.
* Проверка на дубликат и добавление в результат.
* Удаление элемента из соответствующей очереди.
* Добавление новых чисел в очереди.

**3. Вывод результата:**

* Вывести список, содержащий первые n чисел, в разложении которых на простые множители входят только 2, 3 и 5.

**Обоснование выбранных структур и типов данных:**

**1. Очереди (quque):**

* Используются для хранения кандидатов на следующие числа в последовательности.
* Поддерживают порядок элементов (FIFO - First In, First Out), что позволяет легко находить наименьший элемент.
* Обеспечивает эффективное добавление и удаление элементов с обоих концов, что важно для работы алгоритма.

**2. Список (list):**

* Используется для хранения отсортированных чисел.
* Позволяет легко добавлять новые числа в конец списка.
* Обеспечивает последовательный доступ к элементам списка для вывода результата.

**3. Целые числа (int):**

* Используются для хранения значений чисел, генерируемых алгоритмом.
* Обеспечивают возможность выполнения арифметических операций.

**4. Логические значения (bool):**

* Используются для проверок (например, для проверки, нужно ли добавлять новое число в результирующий список или нет).
* Процесс нахождения минимального значения (min): избегаем перебора, поддерживаем порядок, предотвращаем дубликаты.

# Тестирование

При правильном вводе данных (натуральное число) на консоли будет выведен список последовательности чисел:

Изображение выглядит как текст, снимок экрана, Шрифт

Содержимое, созданное искусственным интеллектом, может быть неверным.

При ошибочном вводе (не натуральное число или неправильный ввод) на консоли будет выведена соответствующая ошибка:

Изображение выглядит как текст, снимок экрана, Шрифт

Содержимое, созданное искусственным интеллектом, может быть неверным.

Изображение выглядит как текст, снимок экрана, Шрифт

Содержимое, созданное искусственным интеллектом, может быть неверным.

# Код программы

class Exp(Exception):  
 pass # Исключение для пустой очереди  
  
  
class QueueNode:  
 def \_\_init\_\_(self, value):  
 self.value = value  
 self.next = None  
  
  
class Queue:  
 def \_\_init\_\_(self):  
 self.head = None  
 self.tail = None  
  
 def enqueue(self, value):  
 new\_node = QueueNode(value) # Создаем новый узел  
 if self.tail:  
 self.tail.next = new\_node # Связываем текущий хвост с новым узлом  
 self.tail = new\_node # Перемещаем хвост в конец  
 else:  
 self.head = self.tail = new\_node # Если очередь была пустая  
  
 def dequeue(self):  
 if not self.head:  
 raise Exp("Очередь пуста")  
 value = self.head.value # Берем значение из головы очереди  
 self.head = self.head.next # Сдвигаем голову на следующий элемент  
 if not self.head:  
 self.tail = None # Если очередь опустела — сбрасываем хвост  
 return value  
  
 def peek(self):  
 if not self.head:  
 raise Exp("Очередь пуста")  
 return self.head.value # Возвращаем значение головы без удаления  
  
 def is\_empty(self):  
 return self.head is None  
  
  
class Hamming:  
 def \_\_init\_\_(self):  
 self.x2 = Queue() # Очередь для чисел, кратных 2  
 self.x3 = Queue() # Кратных 3  
 self.x5 = Queue() # Кратных 5  
 self.generated = set() # Множество для контроля уникальных чисел  
  
 def generate(self, n):  
 if not isinstance(n, int) or n <= 0:  
 raise ValueError("Число n должно быть натуральным числом")  
  
 self.x2.enqueue(2)  
 self.x3.enqueue(3)  
 self.x5.enqueue(5)  
 self.generated.update({2, 3, 5}) # Добавляем в множество для проверки уникальности  
  
 result = [1] # Первое число в последовательности  
  
 while len(result) < n:  
 try:  
 candidates = []  
 if not self.x2.is\_empty():  
 candidates.append(self.x2.peek()) # Берём первый элемент из x2  
 if not self.x3.is\_empty():  
 candidates.append(self.x3.peek()) # Из x3  
 if not self.x5.is\_empty():  
 candidates.append(self.x5.peek()) # Из x5  
  
 x = min(candidates) # Выбираем минимальный кандидат  
  
 result.append(x) # Добавляем в результат  
  
 # Удаляем x из соответствующих очередей, если он там есть  
 if not self.x2.is\_empty() and self.x2.peek() == x:  
 self.x2.dequeue()  
 if not self.x3.is\_empty() and self.x3.peek() == x:  
 self.x3.dequeue()  
 if not self.x5.is\_empty() and self.x5.peek() == x:  
 self.x5.dequeue()  
  
 # Добавляем новые числа, умножая x на 2, 3 и 5, если они ещё не были добавлены  
 for multiplier, queue in [(2, self.x2), (3, self.x3), (5, self.x5)]:  
 new\_val = x \* multiplier  
 if new\_val not in self.generated:  
 queue.enqueue(new\_val)  
 self.generated.add(new\_val)  
  
 except Exp as e:  
 print("Ошибка в работе очереди:", e)  
 break  
  
 return result  
  
  
def main():  
 print("Генератор чисел, разлагающихся на множители 2,3,5 (числа Хамминга).")  
 while True:  
 try:  
 n = int(input("Введите количество чисел для генерации (натуральное число): "))  
 generator = Hamming()  
 numbers = generator.generate(n)  
 print("Последовательность:")  
 print(\*numbers)  
 break  
 except ValueError as ve:  
 print("Ошибка ввода:", ve)  
 except Exception as e:  
 print("Произошла ошибка:", e)  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()