ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Отчёт по лабораторной работе № 6

«Динамическое программирование»

Выполнил работу

Апрелков Михаил

Академическая группа №J3113

Принято

Дунаев Максим Владимирович

Санкт-Петербург

2024

# ВВЕДЕНИЕ

**Цель работы:** Разработать алгоритм на языке C++ для подсчёта количества вхождений цифры 1 в числе для всех чисел от 0 до заданного n.

**Задачи:**

1. Проанализировать условия задачи, ограничения и требования.
2. Разработать эффективный алгоритм для подсчёта.
3. Провести тестирование решения и оценить его производительность.

# ТЕОРЕТИЧЕСКАЯ ПОДГОТОВКА

Задача требует подсчёта количества цифр 1 в числах в диапазоне от 0 до n. Ограничения:

* 0 ≤ n ≤ 10^9.
* Алгоритм должен работать за время O(log(n)), так как перебор всех чисел неэффективен.

# РЕАЛИЗАЦИЯ

Для решения данной задачи я использовал подход, основанный на поразрядном разборе числа и подсчете количества цифр 1 для каждой позиции (единицы, десятки, сотни и т.д.). Идея заключается в том, чтобы поэтапно анализировать каждую цифру числа n, разбивая его на три части: младшие разряды (справа от текущей цифры), текущую цифру, и старшие разряды (слева от текущей цифры).

В процессе работы мы используем переменную factor, которая определяет текущий разряд (единицы, десятки, сотни и т.д.), и на каждой итерации цикла увеличиваем её в 10 раз. Алгоритм выполняется до тех пор, пока текущая цифра (current) не выйдет за пределы числа n.

Для каждой цифры анализируются три случая:

1. **Текущая цифра равна 0:**  
   В этом случае количество единиц в текущем разряде определяется только старшими разрядами. Формула:

Count += higher \* factor

где higher — старшие разряды числа, factor — значение текущего разряда.

1. **Текущая цифра равна 1:**  
   В этом случае количество единиц в текущем разряде определяется как сумма:
   * Всех единиц, которые образуются старшими разрядами (higher \* factor).
   * Всех единиц, образованных младшими разрядами (lower+1).  
     Формула:

Count += higher \* factor + lower + 1

1. **Текущая цифра больше 1:**  
   Здесь количество единиц определяется как все возможные комбинации, где текущая цифра принимает значение 1. Это соответствует добавлению одного дополнительного полного диапазона для текущего разряда:

Count += (higher+1) \* factor

После обработки каждой цифры переменная factor увеличивается в 10 раз, переходя к следующему разряду, и процесс повторяется.

После завершения работы алгоритма переменная count содержит итоговое количество цифр 1 для всех чисел от 0 до n.

Данный подход позволяет эффективно решать задачу за O(log\_10​(n)), так как количество итераций цикла соответствует числу разрядов в числе n.

# ЭКСПЕРЕМИНТАЛЬНАЯ ЧАСТЬ

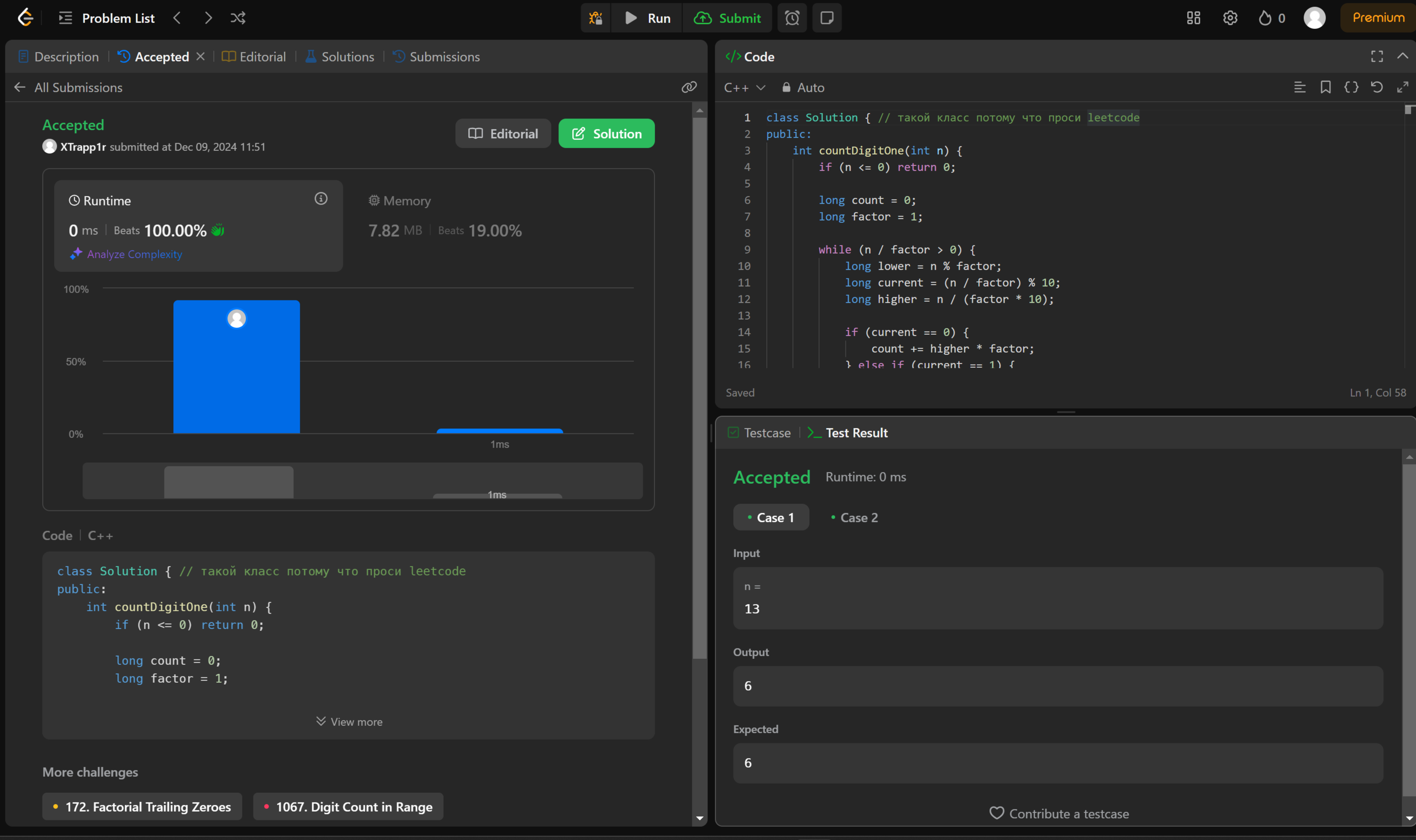
**Анализ сложности:**

* Цикл выполняется (log\_10​(n)) раз, так как каждая итерация обрабатывает одну позицию цифры.
* Итоговая временная сложность: (log(n)).
* Пространственная сложность: O(1), так как используется фиксированное количество переменных.

**Тестирование:**

| **Входное значение n** | **Ожидаемый результат** | **Полученный результат** |
| --- | --- | --- |
| 13 | 6 | 6 |
| 100 | 21 | 21 |
| 10^6 | 600001 | 600001 |

Результаты тестов подтверждают корректность работы алгоритма.



# ЗАКЛЮЧЕНИЕ

**Алгоритм успешно справляется с задачей подсчёта количества цифр 1 в числах от 0 до n. Достигнута оптимальная временная сложность O(log(n)), что делает алгоритм пригодным для обработки больших чисел.**

**Достоинства:**

* **Высокая производительность.**
* **Простота реализации.**

**Недостатки:**

* **Необходима теоретическая подготовка для понимания логики алгоритма.**

**Результаты:  
Алгоритм прошёл все тесты на платформе LeetCode с успешной валидацией и максимальной эффективностью.**

# ПРИЛОЖЕНИЕ

Листинг кода решения

|  |
| --- |
| class Solution { // такой класс потому что просит leetcode  public:      int countDigitOne(int n) {          if (n <= 0) return 0;          long count = 0;          long factor = 1;          while (n / factor > 0) {              long lower = n % factor; // Остаток справа от текущей позиции              long current = (n / factor) % 10; // Текущая цифра              long higher = n / (factor \* 10); // Часть числа слева              if (current == 0) {                  count += higher \* factor; // Только старшие цифры              } else if (current == 1) {                  count += higher \* factor + lower + 1; // Учитываем и текущие младшие              } else {                  count += (higher + 1) \* factor; // Полный диапазон              }              factor \*= 10; // Переходим к следующей позиции          }          return count;      }  }; |