Московский авиационный институт (национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №7 по курсу «Дискретный анализ»

Студент: А. М. Голев
Преподаватель: С. А. Михайлова
Группа: М8О-301Б

Дата: 19.10.25
Оценка:
Подпись:

Москва, 2025

Лабораторная работа №7

Задача: Задано целое число n. Необходимо найти количество натуральных (без нуля) чисел, которые меньше n по значению и меньше n лексикографически (если сравнивать два числа как строки), а так же делятся на m.

Формат ввода: `В первой строке строке задано` $1 \leq n \leq 10^{18}$ `и` $1 \leq m \leq 10^5$

Формат вывода: `Необходимо вывести количество искомых чисел.`

1 Описание

Требуется написать реализацию алгоритма для подсчета количество чисел, арифметически и лексикографически меньших данного числа n, кратных данному числу m. Для решения задачи воспользуемся методом динамического программирования.

Динамическое программирование, как и метод «разделяй и властвуй», позволяет решать задачи, комбинируя решения вспомогательных подзадач [1]. Метод основывается на использовании оптимальной подструктуры. Наличие такой подструктуры означает, что оптимальное решение подзадач меньшего размера может быть использовано для решения исходной задачи [2].

Введем следующие обозначения l - количество цифр в числе n, n_i - i-я цифра числа n, $n[:j]$ - число, составленное из цифр числа n от 0-й до j-й невключительно (пусть индексы цифр начинаются с нуля), $f(k)$ - количество чисел, арифметически и лексикографически меньших k, кратных m.

Рассмотрим алгоритм решения задачи для случая $m = 1$:

1. Вычислить $f(n[:1]) = n[:1] - 1$.
2. Для каждого k от 1 до l - 1 вычислить $f(n[:k{+}1]) = f(n[:k])+1+n[:k{+}1]-10^k-1$ (для наглядности формулы, подобные члены не приведены).
3. $f(n[:l-1])$ - искомое количество чисел.

Докажем корректность данного алгоритма. Т. к. любое число кратно 1, ответом будет количество чисел арифметически и лексикографически меньших n.

Теорема 1: количество чисел, арифметически и лексикографически меньших k, длина которых равна длине k, вычисляется по формуле $k - 10^{d-1} - 1$, где d - количество длина числа k (количество цифр).

Доказательство: По определению десятичной системы счисления, все числа, длина которых равна d, приадлежат отрезку $[10^{d-1}, 10^d - 1]$. Пусть длина числа x равна длине числа k и все цифры от 0 до $i - 1$ в этих числах совпадают. Если $x_i > k_i$, то $x > k$ алгебраически (по определению позиционных систем счисления) и лексикографически (все предшествующие цифры равны) вне зависимости от следующих за i-й цифр. Если $x_i < k_i$, то $x < k$ алгебраически и лексикографически вне зависимости от следующих за i-й цифр. Таким образом, для того, чтобы x было лексикографически не превосходило k необходимо и достаточно, чтобы оно не превосходило k алгебраически. Следовательно, для того, чтобы некоторое число y имело длину d и лексикографически (и алгебраически) не превосходило k достаточно, чтобы оно принадлежало $[10^{d-1}, 10^d - 1] \cap [0, k] = [10^{d-1}, k]$. Требуется, чтобы y было строго меньше k, поэтому исключим k из этого множества: $[10^{d-1}, k)$. количество натуральных чисел в нем можно вычислить по формуле $k - 10^{d-1} - 1$.

Теорема 2: числа, входящие в $f(n[:k])$, также входят в $f(n[:k+1])$.

Доказательство: числа в $f(n[:k])$ меньше $n[:k]$ и $n[:k] \leq n[:k+1]$, т. к. содержит меньше цифр. Следовательно числа в $f(n[:k])$ меньше $n[:k+1]$. Числа в $f(n[:k])$ лексикографически меньше $n[:k]$ и $n[:k]$ лексикографически меньше $n[:k+1]$ (т. к. является префиксом). Следовательно числа в $f(n[:k])$ лексикографически меньше $n[:k+1]$. Таким образом, числа, входящие в $f(n[:k])$, также входят в $f(n[:k+1])$.

Следствие 1: $f(n[:k])$ содержит все числа, входящие в $f(n[:k+1])$, длина которых от 1 до k, кроме самого числа n[:k].

Доказательство: Пусть существует число w с длиной $1 \leq d \leq k$, входящее в $f(n[:k+1])$ и не входящее в $f(n[:k])$. Тогда это число лексикографически и алгебраически меньше числа $n[:d+1]$ (т. к. является его префиксом). Значит w входит в $f(n[:d+1])$, тогда по теореме 2 оно входит в $f(n[:d+2])$, $f(n[:d+3])$, ... $f(n[:k])$. Получили противоречие, следовательно такого числа не существует. Это следствие обосновывает существование оптимальной подструктуры.

Теорема 3: $f(n[:k+1]) = f(n[:k]) + 1 + n[:k+1] - 10^k - 1$.

Доказательство: Все числа, длины которых от 1 до k учитываются в $f(n[:k])$ (верно из следствия 1). Искоючение составляет число $n[:k]$. Чтобы его учесть, прибавляется 1. Числа, длины которых равны k + 1, учитываются в $n[:k+1] - 10^k - 1$ (верно из теоремы 1). Числа, длина которых больше $k+1$ будут больше числа $n[:k+1]$ и не войдут в ответ.

Таким образом алгоритм корректно решает задачу для $m = 1$. Для решения задачи при произвольном значении m необходимо исключить числа, не кратные m. Вместо безусловного прибавления 1 для учета числа $n[:k]$, прибавление осуществляется для $n[:k]$, кратных m. Вместо прибавления количества всех натуральных чисел, принадлежащих $[10^{d-1}, k)$, выполняется прибавление количества чисел, принадлежащих этому отрезку и не кратных m, которое также может быть вычислено за $O(1)$.

Таким образом алгоритм вычисляет искомое число за время $O(l) = O(log(n))$.

2 Исходный код

Программа определяет функцию $inRange$, вычисляющую количество чисел, принадлежащих $[from, to)$, кратных m. Для этого вычисляется, наименьшее число $lower$, кратное m, большее или равное $from$, и наибольшее число кратное $upper$, кратное m, меньшее $from$. Если интервал пустой, возвращается 0. В противном случае, количество чисел, кратных m вычисляется по формуле $(upper - lower)/m + 1$.

Далее следует функция $main$ - точка входа в программу. Число n считывается в виде строки для удобства получения отдельных его цифр. Вместе с ним считывается m в виде числа. Вычисляется длина m.

Вводится массив dp для хранения значений $f(n[:k])$. Вводятся переменные $before$ (хранит $n[:k]$), $from$ (хранит 10^k) и to (хранит $n[:k+1]$).

Вычисляется $dp[0]$. Далее следует цикл, вычисляющий $f(n[:k])$ для k от одного до l. На каждом шаге k значения $from$, to и $before$ вычисляются за константный набор действий из предыдущего результата, что также можно считать использованием метода динамического программирования.

Далее следует вычисление $dp[k+1]$. Для этого к значению $dp[k+1]$ прибавляется количество чисел, принадлежащих $[from, to)$ и кратных m (вычисляется при помощи функции $inRange$). Для учета $n[:k]$ значение $before$ проверяется на кратность m, и, в случае успеха, значение $f(n[:k+1])$ увеличивается на 1.

Наконец, после завершения цикла, программа выводит значение $dp[l-1]$ в качестве ответа.

```c
#include <stdio.h>
#include <inttypes.h>
#include <string.h>

uint64_t inRange(uint64_t from, uint64_t to, uint64_t m) {
    uint64_t lower = (from / m) * m;
    if (from % m != 0) lower += m;

    uint64_t upper = ((to - 1) / m) * m;

    if (upper < lower) return 0;

    return (upper - lower) / m + 1;
}

int main() {
    char nStr[22];
    uint32_t m;

    scanf("%s %d", nStr, &m);
    int length = strlen(nStr);
```

```
22
23      uint64_t dp[22];
24
25      uint64_t before;
26      uint64_t from = 1;
27      uint64_t to = nStr[0] - '0';
28
29      dp[0] = inRange(from, to, m);
30
31      for (int i = 1; i < length; i++) {
32          from *= 10;
33
34          before = to;
35          to = to * 10 + (nStr[i] - '0');
36
37          dp[i] = dp[i - 1] + inRange(from, to, m);
38          if (before % m == 0) dp[i]++;
39      }
40
41      printf("%ld\n", dp[length - 1]);
42
43      return 0;
44  }
```

3 Консоль

```
gcc main.c -o app.out
./app.out
42 3
11
```

4 Тест производительности

Тест производительности представляет из себя следующее: сравнивается время решения задачи переборным алгоритмом и алгоритмом, использующим метод динамического программирования. В тесте $n = 98765456789$ и $m = 343$.

```
smoking_elk@DESKTOP-PJPQAEE:~/discran-labs/lab7/benchmark$ make
gcc straight.c -o straight.out
gcc dp.c -o dp.out
smoking_elk@DESKTOP-PJPQAEE:~/discran-labs/lab7/benchmark$ make test_straight
./straight.out <./test.txt
time: 11979.960000ms
287546020
smoking_elk@DESKTOP-PJPQAEE:~/discran-labs/lab7/benchmark$ make test_dp
./dp.out <./test.txt
287546020
time: 0.026000ms
```

Как видно, из результатов, алгоритм, использущий метод динамического программирования, показывает колоссальное ускорение, по сравнению с алгоритмом, использующим метод перебора. Этот результат коррелирует с различием в сложностных оценках: $O(n * log(n))$ против $O(log(n))$.

5 Выводы

Выполнив седьмую лабораторную работу по курсу «Дискретный анализ», я научился использовать метод динамического программирования, находить и обосновывать существование оптимальной подструктуры. Также я научился использовать модуль $< inttypes.h >$ для наглядной записи числовых типов требуемого размера.

Список литературы

[1] Томас Х. Кормен, Чарльз И. Лейзерсон, Рональд Л. Ривест, Клиффорд Штайн. *Алгоритмы: построение и анализ, 2-е издание.* — Издательский дом «Вильямс», 2007. Перевод с английского: И. В. Красиков, Н. А. Орехова, В. Н. Романов. — 1296 с. (ISBN 5-8459-0857-4 (рус.))

[2] *Динамическое программирование — Википедия.*
URL: `https://ru.wikipedia.org/wiki/Динамическое_программирование` (дата обращения: 12.10.2025).