Лабораторная работа № 7 по курсу дискретного анализа: «Динамическое программирование»

Выполнил студент группы М8О-307Б-21: Меркулов Фёдор Алексеевич

Условие:

Вариант: 3 Количество чисел

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 без остатка.

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.

Пример

Ввод	Вывод
42 3	11

Метод решения

Для решения поставленной задачи воспользуемся методом динамического программирования.

Динамическое программирование (DP) - это метод решения задач, который использует подход "разделяй и властвуй". Основная идея DP заключается в том, что сложная задача разбивается на более простые подзадачи, которые решаются независимо, и затем результаты комбинируются для нахождения ответа на исходную задачу.

Так как количество чисел кратных $m, \leq n$ можно легко посчитать с помощью операции целочисленного деления на m, то основным разбиением на более простые подзадачи будет нахождение количества чисел лексикографически меньших n.

Если представлять n как строку T, то каждая подстрока $T[1 \dots i]$, $i < T.size()$, является лексикографически меньше строки T, при $i = T.size()$, подстрока будет просто равна T, хотя по условию и требуется чтобы все числа были $< n$, можно рассмотреть и само n, прописав дополнительное условие, которое будет

отслеживать является ли n кратным m, так как только в этом случае ответ будет на 1 больше и поэтому мы просто уменьшим ответ на 1, если это условие выполняется.

Итак, использование динамического программирования в этой задаче заключается в выделении вектора a размером $T.size()$ и хранение в i позиции $(i = \overline{0, T.size() - 1})$ подстрок $T[0 \dots i]$, представленных в виде чисел, тем самым в i позициях будут храниться максимальные числа, лексикографически меньшие n, имеющие $i + 1$ символ ($< 10^{i+1}$), далее мы просто для каждой позиции посчитаем количество чисел кратных m, и имеющих ровно $i + 1$ символ, то есть просто сделаем $a[i] = \frac{a[i]}{m} - \frac{(10^i - 1)}{m}$, где $\frac{a[i]}{m}$ – количество чисел кратных m, $\leq a[i]$ (количество чисел кратных m, лексикографически меньших n, имеющих не более $i + 1$ символ); $\frac{(10^i - 1)}{m}$ – количество чисел кратных m, $\leq 10^i - 1$ (количество чисел кратных m, имеющих не более i символ), тем самым теперь в $a[i]$ будет храниться количество чисел кратных m, лексикографически меньших n и имеющих ровно $i + 1$ символ, сложив все такие числа, мы получим количество искомых чисел.

Описание программы

Считываются входные данные, заполняется вектор a размером с длину числа n, заполняется максимальными числами, лексикографически меньшими n, имеющими $i + 1$ символ, затем вектор проходится ещё раз и с помощью операции $a[i] = \frac{a[i]}{m} - \frac{(10^i - 1)}{m}$, начинает хранить в каждой позиции количество чисел кратных m, лексикографически меньших n и имеющих ровно $i + 1$ символ, складываем все значения хранящиеся в i позициях, проверяем не является ли число n кратным m и выводим ответ.

Дневник отладки

$WA16$ из-за неправильной логики (сразу же вычитал $10^i - 1$ в каждом $a[i]$, а только потом делил на m)
$WA32$ из-за определения длины числа n с помощью $log10(n)$ (Начал переводить n в строку $stringN$ и просто определял длину строки)

Тест производительности

Время работы алгоритма я решил проверить с помощью утилиты time. Тесты имеют название $testN.txt$, где N означает количество 9 в n (т,к. $1 \leq n \leq 10^{18}$, то при $N = 18,\ n = 10^{18}$).

papik@papik-VirtualBox:~$ time ./DA7 < test1.txt

real 0m0,007s

user 0m0,000s

```
sys	0m0,007s
papik@papik-VirtualBox:~$ time ./DA7 < test2.txt
real	0m0,007s
user	0m0,006s
sys	0m0,001s
papik@papik-VirtualBox:~$ time ./DA7 < test3.txt
real	0m0,007s
user	0m0,003s
sys	0m0,004s
papik@papik-VirtualBox:~$ time ./DA7 < test4.txt
real	0m0,007s
user	0m0,000s
sys	0m0,007s
papik@papik-VirtualBox:~$ time ./DA7 < test5.txt
real	0m0,007s
user	0m0,001s
sys	0m0,006s
papik@papik-VirtualBox:~$ time ./DA7 < test6.txt
real	0m0,007s
user	0m0,006s
sys	0m0,001s
papik@papik-VirtualBox:~$ time ./DA7 < test7.txt
real	0m0,007s
user	0m0,000s
sys	0m0,007s
papik@papik-VirtualBox:~$ time ./DA7 < test8.txt
real	0m0,007s
user	0m0,000s
sys	0m0,007s
papik@papik-VirtualBox:~$ time ./DA7 < test9.txt
real	0m0,007s
user	0m0,001s
sys	0m0,006s
papik@papik-VirtualBox:~$ time ./DA7 < test10.txt
```

real 0m0,007s

user 0m0,000s

sys 0m0,007s

papik@papik-VirtualBox:~$ time ./DA7 < test11.txt

real 0m0,007s

user 0m0,000s

sys 0m0,007s

papik@papik-VirtualBox:~$ time ./DA7 < test12.txt

real 0m0,007s

user 0m0,001s

sys 0m0,006s

papik@papik-VirtualBox:~$ time ./DA7 < test13.txt

real 0m0,007s

user 0m0,006s

sys 0m0,001s

papik@papik-VirtualBox:~$ time ./DA7 < test14.txt

real 0m0,007s

user 0m0,001s

sys 0m0,006s

papik@papik-VirtualBox:~$ time ./DA7 < test15.txt

real 0m0,007s

user 0m0,000s

sys 0m0,007s

papik@papik-VirtualBox:~$ time ./DA7 < test16.txt

real 0m0,007s

user 0m0,004s

sys 0m0,003s

papik@papik-VirtualBox:~$ time ./DA7 < test17.txt

real 0m0,007s

user 0m0,000s

sys 0m0,007s

papik@papik-VirtualBox:~$ time ./DA7 < test18.txt

real 0m0,007s

user 0m0,006s

sys 0m0,001s

Согласно данным измерениям, можно сказать, что с ростом количества входных данных время работы согласуется с предполагаемой логарифмической сложностью (3 раза проходим по вектору размером $log_{10}n$, то есть сложность $O(log_{10}n)$, а так как $1 \leq n \leq 10^{18}$ по условию, то $\theta(log_{10}10^{18}) = \theta(18)$, то есть на таковых входных данных алгоритм работает за константу)

Вывод

Выполнив лабораторную работу №7 по курсу «Дискретный анализ», я изучил динамическое программирование и убедился, что оно помогает в оптимизации решения задачи, благодаря разбиению задачи и решению более простых подзадач.