Московский авиационный институт
(национальный исследовательский университет)

Факультет компьютерных наук и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №7 по курсу «Дискретный анализ»

Студент: С. Ю. Свиридов
Группа: М8О-306Б-22
Дата:
Оценка:
Подпись:

Москва 2024

Динамическое программирование

Задача: С. 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.

1 Описание

Для решения этой задачи прибегнем к методам динамического программирования. Динамическое программирование - это система решения заданий, которая предполагает, что большая проблема будет разбита на более мелкие задачи, которые более понятные в решении (способ решения сложных задач путём разбиения их на более простые подзадачи). Задачу можно решить и наивным методом, однако это будет крайне неэффективно в сравнении с "динамическим решением". Идея моего метода в том, чтобы разбивать число n на диапазоны. Количество диапазонов будет пропорционально длине числа n. Для каждого такого диапазона будем подсчитывать количество подходящих чисел и в конце получим искомую сумму.

Реализация

Для начала нужно понять делится ли n на m без остатка. Если да, то итоговое количество чисел нужно сразу уменьшить на единицу, чтобы исключить m. Далее подсчитываем длину числа n и запускаем цикл. Для каждого диапазона нижним пределом будет служить число, равное $pow(10, len(n) - 1)$; верхним диапазоном будет выступать само число n, которое на каждой итерации будет уменьшаться на один разряд. Для текущей итерации количество подходящих чисел рассчитывается по формуле $n//m - lower\_limit//m$, где $lower\_limit$ - нижний предел текущей итерации. Существует так же ряд ограничений и проверок: Например, при вводе n, кратном 10, алгоритм сразу же завершается и выводит 0, чтобы не делать лишнюю работу. Так же второе условие на проверку кратности текущего нижнего предела диапазона значению m и его сравнение с нулем. Если он не кратен m или равен 0, то все оставляем как было, иначе уменьшаем на единицу. Это связано с тем, чтобы не добавлять ненужных чисел в нашу итоговую сумму

2 Исходный код

Приложен исходный код программы.

```
#include <iostream>
#include <cmath>

unsigned int len(unsigned long long n) {
    unsigned int l = 0;
    while (n) {
        n /= 10;
        l++;
    }
```

```
    return l;
}

long long count(unsigned long long n, unsigned long long m) {
    long long count;

    if (n % m == 0) {
        count = -1;
    }
    else {
        count = 0;
    }

    while (n != 0) {
        if (len(n) != len(n - 1)) {
            break;
        }
        unsigned long long limit = pow(10, len(n) - 1);
        if (limit % m != 0 || limit == 0) {
            limit = limit;
        }
        else {
            limit--;
        }
        count += n / m - limit / m;
        n /= 10;
    }
    return count;
}

int main () {
    unsigned long long n, m;
    long long res = 0;
    std::cin >> n >> m;
    res = count(n, m);
    if (res < 0) {
        std::cout << 0 << std::endl;
    }
    else {
        std::cout << res << std::endl;
    }
    return 0;
}
```

3 Консоль

```
potatogrill24@DESKTOP-7CM71EV:~/progs/Diskran/laba7$ ./a.out
```

```
42 3
11
potatogrill24@DESKTOP-7CM71EV:~/progs/Diskran/laba7$ ./a.out
9994912859122999591 1212
8246161403669782
potatogrill24@DESKTOP-7CM71EV:~/progs/Diskran/laba7$ ./a.out
12421421 1
2690472
potatogrill24@DESKTOP-7CM71EV:~/progs/Diskran/laba7$ ./a.out
1000000000000 1
0
```

Сложность алгоритма - $O(log(len(n))$

4 Тест производительности

В тестах я решил произвести сравнение времени исполнения наивного алгоритма и алгоритма, основанного на динамических методах.

```
potatogrill24@DESKTOP-7CM71EV:~/progs/Diskran/laba7$ g++ bench.cpp
potatogrill24@DESKTOP-7CM71EV:~/progs/Diskran/laba7$ ./a.out
100000000000 1
Naive: 0 ms
Not Naive: 0 ms
potatogrill24@DESKTOP-7CM71EV:~/progs/Diskran/laba7$ ./a.out
0 1112
Naive: 0 ms
Not Naive: 0 ms
potatogrill24@DESKTOP-7CM71EV:~/progs/Diskran/laba7$ ./a.out
12345 2
Naive: 486 ms
Not Naive: 14 ms
potatogrill24@DESKTOP-7CM71EV:~/progs/Diskran/laba7$ ./a.out
124214124 126
Naive: 72310 ms
Not Naive: 19 ms
potatogrill24@DESKTOP-7CM71EV:~/progs/Diskran/laba7$ ./a.out
161262212512 1242
Naive: 10556019 ms
Not Naive: 30 ms
```

Как видно из тестов, наивный алгоритм намного медленнее, потому что в его конструкции лежит функция $std :: to\_string()$, которая переводит число в строку. Из-за нее алгоритм имеет очень высокую сложность работы и выполняется в десятки или сотни раз медленнее.

5 Выводы

В ходе выполнения данной лабораторной работы я познакомился с динамическими методами программирования. Я узнал, насколько они могут быть эффективными в сравнении с обычными способами решения задач и в то же время простыми в реализации.

Список литературы

[1] *Динамическое программирование*
URL: https://habr.com/ru/articles/777618/

[2] *Динамическое программирование*
URL: https://neerc.ifmo.ru/wiki/index.php?title=Динамическое_программирование