Лабораторная работа № 7 по курсу дискретного анализа: Динамическое программирование

Выполнил студент группы М8О-208Б-20 *Ядров Артем.*

Условие

Кратко описывается задача:

1. При помощи метода динамического программирования разработать алгоритм решения задачи, определяемой своим вариантом
2. *Вариант:* Количество чисел
3. *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 также делятся на m без остатка.

Метод решения

Динамическое программирование - метод решения задач, при котором сложная задача разбивается на более простые, решение сложной задачи составляется из решений простых задач.

Попытаемся применить этот метод к нашей задаче. Очевидно, что операция лексиграфического сравнения довольно затратна, поэтому попытаемся избавиться от нее: рассмотрим произвольное число вида: $a_1a_2a_3a_4...a_n$.

Заметим, что длина числа, удовлетворяющего условию может варьироваться в пределах от 1 до n. Таким образом, можно разбить нашу задачу на n одинаковых подзадач.

Подзадачи формируются следующим образом: пусть на вход задана фиксированная длина k числа , которое должно удовлетворять условию. В таком случае должна выполняться следующая система условий.

$$\begin{cases} b_1 < a_1 \\ b_2 < a_2 \\ ... \\ b_k < a_k \end{cases}$$

К числу $b_1b_2....b_k$ можно дописать нули справа и тогда такое число будет меньше n по значению и лексикографически. Возьмем минимальное число $b_1b_2...b_k$, удовлетворяющее системе: очевидно, что оно имеет ведущую единицу и $k-1$ нулей. Этим числом является 10^{k-1}. Назовем его l. Приступим к проверке второго условия. Увеличим наше число l до числа $Left$ таким образом, что $Left \quad \% \quad m = 0$ и $Left$ - минимальное. Очевидно, что если $l \quad \% \quad m = 0$, то $Left = l$. Иначе $Left = l + m - l \quad \% \quad m$. Также уменьшим нашу "подстроку"исходного числа таким образом, чтобы полученное число $Right$ делилось на m. Тут все очевидно, просто вычтем из числа его остаток от деления на m. Таким образом, мы получили отрезок $[Left; Right]$, где границы делятся на заданное число m. И выполняется следующее условие: $\forall x \in [Left; Right] \Rightarrow x < n[0 : k]$ (и по значению, и лекскиографически)

Осталось определить, сколько чисел внутри этого отрезка делятся нацело на m. Очевидно, что их $(Right - Left)/m + 1$, т.к. каждое m-ое число делится на m и границы делятся на m. Итак, мы получили ответ для подзадачи с параметром k.

Ответом на задачу будет сумма по всевозможным k. $Ans = \sum_{k=1}^{length(n)}$, где $length(n)$ - кол-во цифр в записи числа n. То есть это целая часть числа $lg(n)$.

Несложно догадаться, что каждую подзадачу мы умеем решать за $O(1)$, при условии, что все входные параметры нам даны (не надо вычислять 10^{k-1}, в частности). Тогда всю задачу мы решим за $O(lg(n))$, где $lg(n)$ - десятичный логарифм (логарифм по основанию 10) числа n.

Исходный код

Несмотря на математические выкладки, реализация алгоритма получилась довольно компактной.

```
#include <iostream>
#include <chrono>
long long DP(const long long &r, const long long &l, const long long &m) {
  if (r < l) {
        return 0;
  }
  long long right = r - r % m;
  long long left, ost = l % m;
  if (ost == 0){
        left = l;
  } else{
        left = l + m - ost;
  }
  if (left > right or right <= 0) {
        return 0;
  }
  return (right - left) / m + 1;
}

int main() {
  std::string N;
  long long n = 0, m, tenPow = 1, ans = 0;
```

```
  std::cin >> N >> m;
  for (char i : N) {
        n = n * 10 + i - '0';
        ans += DP(n, tenPow, m);
        tenPow *= 10;
  }
  std::cout << ans - (n % m == 0) << std::endl;
  return 0;
}
```

Дневник отладки

Во время реализации я столкнулся с небольшими проблемами:

1. Проблема с алгоритмом. Изначально для подзадач я брал следующий срез числа: $n[len(n) - k; len(n)]$ или же просто брал правые k чисел. Очевидно, что это неправильно

2. Проблема со временем. Также перейдя к нисходящей концепции я решил задачу рекурсивно. Хотя задача и успешно была проверена на чекере, рекурсия тратит значительно больше времени, нежели итерация.

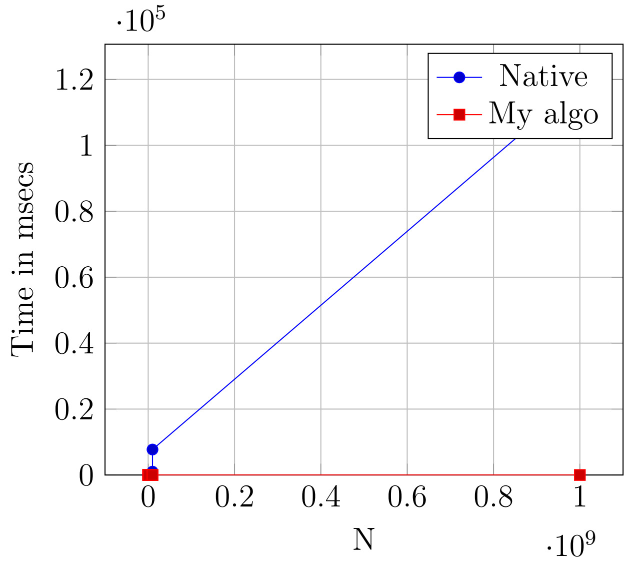
3. Ну и вишенка на торте в виде проблем с вычислениями. Первые две проблемы можно было легко отловить и отладить, когда как эту проблему крайне сложно отследить. К счастью, методом пристального взгляда мне удалось найти слабые места программы.

Тест производительности

При первом прочтении задания у меня возникла идея наивного решения за $O(\frac{nlog(n)}{m})$. Но очевидно, она уступает моему решению. Идея была проста до невозможности: будем идти по всем числам, кратным m, начиная с самого m, до тех пор, пока наши числа меньше n. Таким образом, два из трех условий выполняются. Лексикографическую проверку пришлось бы проводить путем сравнения всех символов в строках.

Рассмотрим худшее время работы наивного алгоритма: очевидно, что оно достигается при $m = 1$. Так как сложность моего алгоритма не зависит от m (сложность операции взятия остатка я принял за $O(1)$), то для него это будет просто средним временем работы.

В качестве тестовых данных будем брать $n = 10^q - 1$ и $m = 1$. Попытавшись прогнать наивный алгоритм для $q > 9$, я столкнулся с бесконечно большим временем выполнения программы. Поэтому будем сравнивать алгоритмы для $q = 1, 2, ..., 9$.



Выводы

В ходе выполнения лабораторной работы я изучил классические задачи динамического программирования и их методы решения, реализовал алгоритм для своего варианта задания.

Также в очередной раз убедился в том, что рекурсия - хорошее средство для ленивого программиста вроде меня, но занимает достаточно много времени. Поэтому если есть возможность пользоваться итерацией, не следует ей пренебрегать.

Динамическое программирование позволяет разработать точные и относительно быстрые алгоритмы для решения сложных задач, в то время, как решение перебором слишком медленное, а жадный алгоритм не всегда даёт правильный результат.