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: К.С. Саженов
Преподаватель: И. Н. Симахин
Группа: М8О-208Б

Дата:
Оценка:
Подпись:

Москва, 2021

Лабораторная работа №7

Задача: При помощи метода динамического программирования разработать алгоритм решения задачи; оценить время выполнения алгоритма и объем затрачиваемой оперативной памяти.
Вариант задачи: 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.

1 Описание

Требуется решить задачу методом динамического программирования.
`Динамическое программирование - это когда у нас есть задача, которую непонятно как решать, и мы разбиваем ее на меньшие задачи, которые тоже непонятно как решать. (c) A.Кумок` [1]
Динамическое программирование – метод программирования, при котором мы разбиваем исходную задачу на подзадачи и решаем эти "маленькие"подзадачи в какомлибо цикле.
Данный метод применим к задачам с оптимальной подструктурой, выглядящим как набор перекрывающихся подзадач, сложность которых чуть меньше исходной: в этом случае время вычислений можно значительно сократить. Как правило, чтобы решить поставленную задачу, требуется решить отдельные части задачи(подзадачи), после чего объединить решения подзадач в одно общее решение. Часто многие из этих подзадач одинаковые или похожи друг на друга. Подход динамического программирования состоит в том, чтобы решить каждую отдельную задачу только один раз, сократив, тем самым, количество вычислений. Это особенно полезно в случаях, когда число повторяющихся подзадач эксоненциально велико. Этапы построения алгоритма решения задач динамическим программированием:

- Описать структуру оптимального решения.
- Составить рекурсивное решение для нахождения оптимального решения.
- Вычислить значения, соответствующего оптимальному решению, методом восходящего анализа.
- Непосредственное нахождение оптимального решения из полученной на предыдущих этапах информации.

2 Исходный код

Идея решения данной задачи основывается на довольно простом наблюдении: На каждый разряд(в десятичной системе счисления) приходится ровно $n/m-(10^{|str(n)|-1}-1)/m$ чисел, которые меньше n лексикографически и делятся на m, где n – само число, m – число из условия, n/m – сколько в принципе чисел делятся на m(которые меньше n), а $(10^{\lfloor log_{10}(n) \rfloor - 1} - 1)/m$ – кол-во чисел, делящихся на m, но больше n лексикографически(для одного старшего разряда). Причем в данном выражении учитываются и числа меньших разрядов(то есть они тоже вычитаются).
То есть итоговое решение это сумма предыдущего ответа(для предыдущего разряда) и приведенной выше формулы для данного разряда. Необходимо повторять данный алгоритм, пока $n > 0$. Файл `main.cpp`:

```cpp
#include <iostream>
#include <vector>
#include <string>
#include <cmath>
#include <algorithm>


int main() {
    std::int64_t n, m;
    std::cin >> n >> m;

    std::vector<std::int64_t> dp;
    dp.resize(std::to_string(n).size()+1);
    dp[0] = -bool(n%m==0);

    std::int64_t i(1);
    while(n > 0){
        dp[i] = n/m - ((std::int64_t)std::pow(10, std::to_string(n).size()-1) - 1) / m
            + dp[i-1];
        n /= 10;
        ++i;
    }

    std::cout << std::max(dp.back(), (std::int64_t)0) << std::endl;
}
```

3 Консоль

```
sakost@sakost-pc ~/university/2 course/diskran/lab7
$ cmake -S . -B cmake-build-debug
--The C compiler identification is GNU 11.1.0
--The CXX compiler identification is GNU 11.1.0
--Detecting C compiler ABI info
--Detecting C compiler ABI info -done
--Check for working C compiler: /sbin/cc -skipped
--Detecting C compile features
--Detecting C compile features -done
--Detecting CXX compiler ABI info
--Detecting CXX compiler ABI info -done
--Check for working CXX compiler: /sbin/c++ -skipped
--Detecting CXX compile features
--Detecting CXX compile features -done
--Configuring done
--Generating done
--Build files have been written to: /home/sakost/university/2 course/diskran/lab7/cmal
sakost@sakost-pc ~/university/2 course/diskran/lab7
$ cmake --build cmake-build-debug --target lab7
[ 50%] Building CXX object CMakeFiles/lab7.dir/main.cpp.o
[100%] Linking CXX executable lab7
[100%] Built target lab7
sakost@sakost-pc ~/university/2 course/diskran/lab7
$ ./cmake-build-debug/lab7
42 3
11
sakost@sakost-pc ~/university/2 course/diskran/lab7
$
```

4 Тест производительности

Наивное решение:

```
#include <algorithm>
#include <iostream>
#include <string>
#include <vector>

using namespace std;

int main() {
    int n, m;
    cin >> n >> m;
    int count = 0;
    for (long long i(n); i > 0ll; --i) {
        auto s1 = to_string(i);
        auto s2 = to_string(n);
        if (i % m == 0 &&
        lexicographical_compare(s1.begin(), s1.end(), s2.begin(), s2.end())) {
            ++count;
        }
    }
    cout << count << endl;
}
```

Сам замер:

```
sakost@sakost-pc ~/university/2 course/diskran/lab7
$ time ./cmake-build-debug/naive <<EOL
10704020 59
EOL
13260
./cmake-build-debug/naive <<<'10704020 59' 1,86s user 0,00s system 99% cpu
1,865 total
sakost@sakost-pc ~/university/2 course/diskran/lab7
$ time ./cmake-build-debug/lab7 <<EOL
10704020 59
EOL
13260
./cmake-build-debug/lab7 <<<'10704020 59' 0,01s user 0,00s system 9% cpu 0,075
total

sakost@sakost-pc ~/university/2 course/diskran/lab7
$ time ./cmake-build-debug/naive <<EOL
107040200 59
```

```
EOL
132585
./cmake-build-debug/naive <<<'107040200 59' 15,34s user 0,00s system 99% cpu
15,346 total
sakost@sakost-pc ~/university/2 course/diskran/lab7
$ time ./cmake-build-debug/lab7 <<EOL
107040200 59
EOL
132585
./cmake-build-debug/lab7 <<<'107040200 59' 0,01s user 0,00s system 78% cpu
0,009 total
sakost@sakost-pc ~/university/2 course/diskran/lab7
$
```

По такому небольшому бенчмарку сразу видно отставание наивного алгоритма, причем в разы.

5 Выводы

Динамическое программирование – частый метод в решении в, на первый взгляд, не решаемых (быстро) задач. Данный метод позволяет построить радикально ускоренную версию алгоритма. Динамическое программирование может быть применимо практически везде, где есть какие-либо перекрывающиеся подзадачи, которые можно выделить из главной задачи. В частности, как мне кажется, данный подход имеет место быть в сложных параллельных вычислениях. Например вычислении выхода разных нейронных сетей или других алгоритмов машинного обучения(поскольку там много перекрывающихся подзадач). В данной лабораторной работе я также укрепил свои знания в подходе динамического программирования. Стоит упомянуть, что ДП – это скорее подход к построению алгоритмов, а не просто конкретный алгоритм.

Список литературы

[1] *Динамическое программирование*
URL: `https://neerc.ifmo.ru/wiki/index.php?title=Динмаическое_программирование`
(дата обращения 10.10.2021)

[2] 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.))