Алгоритм решения задачи "Делимость и сумма"

Антон Дроздовский

1 Условие задачи

Назовём число счастливым, если оно делится на k и сумма его цифр лежит в интервале $[p, q]$. Подсчитайте, сколько счастливых чисел лежат в интервале $[a, b]$.

Input

Первая строка содержит десятичную запись числа k ($1 \leq k \leq 10^{11}$). Вторая строка — десятичную запись чисел p и q ($1 \leq p, q \leq 99$). Третья — числа a и b ($1 \leq a, b \leq 10^{11}$).

Output

Выведите найденное число. (Пробелов в этой строке быть не должно!)

Пример

lucky.in	lucky.out
11 3 6 11 40	2

2 Описание алгоритма

При решении данной задачи был применен метод динамического программирования. Возникающая динамика основана на четырёх параметрах: разряде числа, сумме цифр числа, остаток от деления числа на k и флаг:

```
1. long long dp[13][100][8999][2];
```

Под флагом 0 содержится количество чисел, имеющих i цифр в своём составе, сумму цифр s и делится на k с остатком r, которые меньше числа, по которое считается количество счастливых чисел. Под флагом 1 сохраняем 1 — совпадение префиксов чисел с исходным числом.

Algorithm 1 Заполнение массива

$dp[\ldots][\ldots][\ldots][\ldots] \leftarrow 0$
$dp[0][0][0][1] \leftarrow 1$
for $i = 1$ to $|B|$ **do** ▷ разряд числа
  for $s = 0$ to q **do** ▷ сумма цифр
    for $r = 0$ to $k - 1$ **do** ▷ остаток от деления
      for $d = 0$ to 9 **do** ▷ новая цифра
        $dp[i][s + d][(10 \cdot r + d)\%k][0] \mathrel{+}= dp[i - 1][s][r][0]$
        if $d \leq B_{i-1}$ **then**
          $dp[i][s + d][(10 \cdot r + d)\%k][d == B_{i-1}] \mathrel{+}= dp[i - 1][s][r][1]$
        end if
      end for
    end for
  end for
end for

Для подсчёта количества счастливых чисел по число B мы должны просуммировать количество чисел, которые нацело делятся на k и сумма цифр которых лежит на промежутке $[p, q]$. То есть:

Algorithm 2 Нахождение кол-ва счастливых чисел по число B

$f(B) \leftarrow 0$
for $s \leftarrow p$ **to** q **do**
  $f(B) \leftarrow f(B) + dp[|B|][s][0][0] + dp[|B|][s][0][1]$
end for

То есть для получения количества счастливых чисел на промежутке $[a, b]$ надо составить два массива для чисел b и $a-1$, найти количество счастливых чисел по каждое число и отнять от $f(b)$ $f(a - 1)$.

При инициализации массива была выделена размерность для остатков в 8999 элементов. Это связано с тем, что максимальное значение, которое может принимать k равно 10^{11}, что означает, что и количество различных остатков равно 10^{11}. Массив нереальных размеров. Но для $k \geq 9000$ достаточно решать задачу в лоб, чтобы вложиться в тесты. Перебор чисел в промежутке $[a, b]$, делящиеся на k, и вычисление суммы цифр числа функцией для последующей проверки на $[p, q]$.

3 Решение задачи на C++

```
#define _CRT_SECURE_NO_WARNINGS

#include <algorithm>
#include <array>
#include <cstdio>
```

```cpp
#include <ios>
#include <iostream>
#include <vector>

std::array<std::array<std::array<std::array<long long, 2>,
    8999>, 100>, 13> adp;
std::array<std::array<std::array<std::array<long long, 2>,
    8999>, 100>, 13> bdp;

int getDigitSum(long long num) {
    int sum = 0;
    while (num != 0) {
        sum += num % 10;
        num /= 10;
    }

    return sum;
}

int main() {
    std::ios_base::sync_with_stdio(false);
    std::cin.tie(nullptr);

    std::freopen("lucky.in", "r", stdin);
    std::freopen("lucky.out", "w", stdout);

    long long k;
    int p, q;
    long long a, b;
    std::cin >> k >> p >> q >> a >> b;

    if (k >= 9000) {
        long long tmp = k;

        while (tmp < a) {
            tmp += k;
        }

        long long counter = 0;

        while (tmp <= b) {
            int digitSum = getDigitSum(tmp);
            if (digitSum >= p && digitSum <= q) {
                ++counter;
            }

            tmp += k;
        }

        std::cout << counter << '\n';
```

```

        return 0;
    }

    adp[0][0][0][1] = 1;
    bdp[0][0][0][1] = 1;

    std::vector<int> aDigits;
    std::vector<int> bDigits;

    --a;

    while (a != 0) {
        aDigits.push_back(a % 10);
        a /= 10;
    }

    std::reverse(aDigits.begin(), aDigits.end());

    while (b != 0) {
        bDigits.push_back(b % 10);
        b /= 10;
    }

    std::reverse(bDigits.begin(), bDigits.end());

    std::size_t bSize = bDigits.size();

    int tmpS = 0;
    long long tmpR = 1;
    int sd;

    for (std::size_t i = 1; i <= bSize; ++i) {
        tmpS += 9;
        for (int s = 0; s <= tmpS && s <= q; ++s) {
            for (int r = 0; r < tmpR && r < k; ++r) {
                for (int d = 0; d < 10; ++d) {
                    sd = s + d;
                    if (sd <= q && sd <= tmpS) {
                        bdp[i][sd][(10 * r + d) % k][0] +=
                            bdp[i - 1][s][r][0];
                        if (d <= bDigits[i - 1]) {
                            bdp[i][sd][(10 * r + d) % k][d
                                == bDigits[i - 1]] += bdp[i -
                                 1][s][r][1];
                        }
                    }
                }
            }
        }
```

```

        tmpR *= 10;
    }

    long long fb = 0;
    for (int s = p; s <= q; ++s) {
        fb += bdp[bSize][s][0][0] + bdp[bSize][s][0][1];
    }

    std::size_t aSize = aDigits.size();

    tmpS = 0;
    tmpR = 1;

    for (std::size_t i = 1; i <= aSize; ++i) {
        tmpS += 9;
        for (int s = 0; s <= tmpS && s <= q; ++s) {
            for (int r = 0; r < tmpR && r < k; ++r) {
                for (int d = 0; d < 10; ++d) {
                    sd = s + d;
                    if (sd <= tmpS && sd <= q) {
                        adp[i][sd][(10 * r + d) % k][0] += adp[i - 1][s][r][0];
                        if (d <= aDigits[i - 1]) {
                            adp[i][sd][(10 * r + d) % k][d == aDigits[i - 1]] += adp[i - 1][s][r][1];
                        }
                    }
                }
            }
        }

        tmpR *= 10;
    }

    long long fa = 0;
    for (int s = p; s <= q; ++s) {
        fa += adp[aSize][s][0][0] + adp[aSize][s][0][1];
    }

    std::cout << fb - fa;

    return 0;
}
```

4 Оценка временной сложности

Оценка временной сложности данного кода может быть проведена путем анализа его структуры и количества итераций в циклах. Давайте рассмотрим каждую часть программы и оценим их сложность.

1. Чтение входных данных и инициализация переменных

Эти операции выполняются за константное время $O(1)$.

2. Проверка особого случая, когда $k \geq 9000$

- Цикл `while (tmp < a)` выполняется за $O\left(\frac{a}{k}\right)$ итераций.
- Цикл `while (tmp <= b)` также выполняется за $O\left(\frac{b-a}{k}\right)$ итераций, то есть можно считать $O\left(\frac{b}{a}\right)$.
- Внутри цикла происходит вызов функции `getDigitSum()`, который работает за время $O(\log_{10}(tmp))$. Однако, так как `tmp` ограничено значением `b`, можно считать, что вызов функции работает за время $O(\log_{10}(b))$.

Итоговая сложность этой части кода составляет $O\left(\frac{b}{k}\log(b)\right)$.

3. Инициализация массивов `adp` и `bdp`

Эта операция выполняется за константное время $O(1)$, так как размеры массивов фиксированы.

4. Подготовка чисел `a` и `b`

- Цикл `while (a != 0)` выполняется за $O(\log_{10}(a))$ итераций.
- Цикл `while (b != 0)` выполняется за $O(\log_{10}(b))$ итераций.
- Затем оба вектора `aDigits` и `bDigits` переворачиваются, что также занимает $O(\log_{10}(a))$ и $O(\log_{10}(b))$ времени соответственно.

Итоговая сложность этой части кода составляет $O(\log(b))$.

5. Цикл для вычисления `bdp`

- Внешний цикл `for` выполняется $bSize$ раз, где $bSize$ - количество цифр числа `b`. $O(\log(b))$
- Внутренние циклы и операции внутри них выполняются за $O(q)$, $O(k)$.

Общая сложность этой части кода составляет $O(qk\log(b))$.

6. Подсчет суммы `fb`

- Цикл `for` выполняется $q - p + 1$ раз.
- Внутри цикла выполняются операции за константное время $O(1)$.

Общая сложность этой части кода составляет $O(q)$.

7. Цикл для вычисления `adp`

Аналогично с пунктом 5. Общая сложность этой части кода составляет $O(qk \log(a))$.

8. Подсчет суммы `fa`

- Цикл `for` выполняется $q - p + 1$ раз.
- Внутри цикла выполняются операции за константное время $O(1)$.

Общая сложность этой части кода составляет $O(q)$.

9. Вывод результата

- Операция вывода результата выполняется за константное время $O(1)$.

Учтываем, что $b \geq a$. Получаем две асимптотики: $O\left(\frac{b}{k} \log(b)\right)$ при $k \geq 9000$ и $O(qk \log(b))$.

5 Оценка сложости по памяти

В решении задачи использовались следующие структуры данных:

- Многомерные статические массивы. Константа.
- Динамические массивы для хранения отдельных цифр чисел $a - 1$ и b используют память, пропорциональную размеру чисел $a - 1$ и b соответственно. Поэтому сложность по памяти для этих векторов можно оценить как $O(\log(a - 1) + \log(b))$.

Итоговая оценка: $O(\log(b))$