Kick Start 2022 - Round A

Analysis: Interesting Integers

For simplicity, let us use L_N to represent the number of digits in an integer N. Note that L_N is actually equal to $\lceil \log_{10}(N+1) \rceil$.

Test Set 1

For the small test set, we can simply enumerate all integers in the range from $\mathbf{A}$ to $\mathbf{B}$ and check if each integer is an interesting integer individually. Determining if an integer is interesting or not takes $O(L_\mathbf{B})$ time (to calculate the sum and product of the digits), where $L_\mathbf{B}$ is the maximum number of digits in all integers in the range.

The overall time complexity is $O((\mathbf{B} - \mathbf{A}) \times L_\mathbf{B})$ or just $O(\mathbf{B} \log_{10} \mathbf{B})$.

Test Set 2

If we have a function $CountInterestingIntegers(N)$ to get the number of interesting integers between 1 and N, then the number of interesting integers between two positive integers $\mathbf{A}$ and $\mathbf{B}$ inclusively can be computed by $CountInterestingIntegers(\mathbf{B}) - CountInterestingIntegers(\mathbf{A} - 1)$. If $N < 1$, $CountInterestingIntegers(N)$ is trivially equal to 0.

The following are two different methods to compute the value of $CountInterestingIntegers(N)$.

Method 1

To compute the return value of $CountInterestingIntegers(N)$, first of all let us define a function $f_1(L, P, S)$ which returns the number of combinations of arbitrary L digits $d_1, d_2, \ldots, d_L$ which satisfy the condition that $P \times \prod_{i=1}^{L} d_i$ is divisible by $S + \sum_{i=1}^{L} d_i$, which can be written as $(P \times \prod_{i=1}^{L} d_i) \% (S + \sum_{i=1}^{L} d_i) = 0$.

Based on the definition above, for $L = 0$, the return value of $f_1(L, P, S)$ is 1 if $P \% S = 0$, and 0 otherwise. For $L > 0$, the return value of $f_1(L, P, S)$ is $\sum_{digit=0}^{9} f_1(L - 1, P \times digit, S + digit)$. In the implementation, we can use top-down dynamic programming to memoize the return values of f_1 with different parameter sets in a hash table to speed up the function calls.

Afterward, we can compute the value of $CountInterestingIntegers(N)$ by recursively visiting each digit of N from left (0, the most significant digit) to right ($L_N - 1$, the least significant digit) to construct integers smaller than N and check the number of interesting integers with function f_1. The following is the pseudo code for $CountInterestingIntegers$:

```
def CountInterestingIntegers(N):
  if N == 0:
    return 0
  count = 0
  for L in (1 to (CountDigits(N) - 1)):
    count += CountInterestingIntegersWithNumberOfDigits(L)

  count += CountInterestingIntegersWithPrefixOfN(N, P=1, S=0, digit_index=0, is_first_digit=True)
  return count

def CountInterestingIntegersWithNumberOfDigits(L):
  count = 0
  for digit in (1 to 9):
    count += f1(L - 1, P=digit, S=digit)
  return count

def CountInterestingIntegersWithPrefixOfN(N, P, S, digit_index, is_first_digit):
  if digit_index == CountDigits(N):
    return 1 if S > 0 and P % S == 0 else 0

  if is_first_digit:
    digit_start = 1
  else:
    digit_start = 0

  count = 0
  for digit in (digit_start to (GetIthDigit(N, digit_index) - 1)):
    count += f1(CountDigits(N) - digit_index - 1, P * digit, S + digit)

  count += CountInterestingIntegersWithPrefixOfN(N,
                                                 P * GetIthDigit(N, digit_index),
                                                 S + GetIthDigit(N, digit_index),
                                                 digit_index + 1, is_first_digit=False)
  return count
```

Let M be the number of digits in the maximum number in all possible ranges from $\mathbf{A}$ to $\mathbf{B}$. The number of possible parameter sets of $f_1(L, P, S)$ is bounded by $M \times (\frac{(M+9-1)!}{(9-1)!M!} + 1) \times (9 \times M)$, where the number of possible products is bounded by the combinations of choosing M digits from $\{1, 2, 3, \ldots, 9\}$ without repetitions, plus 1 (the zero product). Besides, computing the return value of $f_1(L, P, S)$ takes $O(1)$ time with memoization. Therefore, the overall time complexity of $f_1(L, P, S)$ is $O(M \times (\frac{(M+9-1)!}{(9-1)!M!} + 1) \times (9 \times M))$. Since $\mathbf{A}, \mathbf{B} \leq 10^{12}$ from the problem statement, we can derive that $M = 13$ and the number of parameter sets is at most $13 \times 203,491 \times (9 \times 13)$, which is equal to $309,509,811$.

A tighter estimate for the number of possible sets of $f_1(L, P, S)$ is $O(\sum_{L=1}^{M} (\frac{(L+9-1)!}{(9-1)!L!} + 1) \times (9 \times L))$, which is bounded by $52,379,145$ when $M = 13$. In fact, only $188,701$ entries are kept in the memoization of f_1 after solving all the test cases in test set 2, which is much smaller than the estimate.

The function $CountInterestingIntegers(N)$ takes $O(M)$ to get the number of interesting integers from 0 to N with the memoization of f_1. Therefore, the overall time complexity for method 1 is $O(M)$, where M is equal to $\lceil \log_{10}(\mathbf{B} + 1) \rceil$.

Notice that the memoization for the return values of $f_1(L, P, S)$ can be reused in all the test cases. Clearing and rebuilding the memoization for every test case may make the implementation exceed time limit since there are 100 test cases in test set 2.

Optimization for Method 1 with Prime Factorization

Since the value of parameter P passed to $f_1(L, P, S)$ is always a product of digits, it can be written as $P = 2^w \times 3^x \times 5^y \times 7^z$, where $2, 3, 5, 7$ are what we call "interesting prime factors" and w, x, y, z are powers of these prime factors. Moreover, the maximum possible value of parameter S is $9M$. From these two observations, we can find out that if $2^w > 9M$, then $P\%S = 0$ if and only if $(P/2)\%S = 0$ for any $S \leq 9M$. This observation applies to other interesting prime factors as well.

Based on the observation aforementioned, we can further improve the performance by reducing the number of possible parameter sets of $f_1(L, P, S)$ by the idea of capping the power of interesting prime factors. Let us define a helper function $CapInterestingPrimeFactors(P)$ which does the following things:

1. Compute the power of interesting prime factors w, x, y, z of input $P = 2^w \times 3^x \times 5^y \times 7^z$.
2. For each interesting prime factor p, define the power of it as v and compute the new power as the maximum value of v' which satisfies $p^{v'} \leq 9M$ and $v' \leq v$.
3. Construct the new product $P' = 2^{w'} \times 3^{x'} \times 5^{y'} \times 7^{z'}$ and return it.

Afterwards, we can replace all the function calls $f_1(L, P, S)$ with $f_1(L, CapInterestingPrimeFactors(P), S)$. Since the maximum possible value of $S = 9 \times 13 = 117$ in test set 2 (this is just an upper bound based on the definition above), the maximum value of P' after capping is $2^6 \times 3^4 \times 5^2 \times 7^2$, and the total number of possible values of $CapInterestingPrimeFactors(P)$ is $7 \times 5 \times 3 \times 3 = 315$. Therefore, the number of parameter sets of the improved $f_1(L, P', S)$ is at most $13 \times 315 \times (9 \times 13) = 479,115$ which is a huge improvement over the pre-optimized method 1. In fact, only 56,853 entries are kept in the memoization of f_1 after solving all the test cases in test set 2.

Method 2

Method 2 is similar to method 1 but we set a fixed target digit sum when computing the number of interesting integers, then add up the number of interesting integers with all possible digit sums.

Let us define another function $f_2(L, P, S, S_{target})$ which returns the number of combinations of arbitrary L digits $d_1, d_2, \ldots, d_L$ that satisfy the conditions:

- $S + \sum_{i=1}^{L} d_i = S_{target}$
- $P \times \prod_{i=1}^{L} d_i$ is divisible by S_{target} ($(P \times \prod_{i=1}^{L} d_i)\%S_{target} = 0$)

Notice that $f_2(L, P, S, S_{target}) = f_2(L, P\%S_{target}, S, S_{target})$. For $L = 0$, the return value is 1 if $S = S_{target}$ and $P\%S_{target} = 0$, and 0 otherwise. For $L > 0$, the return value of $f_2(L, P, S, S_{target})$ is equal to $\sum_{digit=0}^{9} f_2(L - 1, (P \times digit)\%S_{target}, S + digit, S_{target})$.

In the implementation, we can pre-compute return values of f_2 with all possible parameter sets and store the values in a 4-dimensional array or hash map. Let M be the number of digits in the maximum number in all possible ranges from $\mathbf{A}$ to $\mathbf{B}$, the possible digits sum ranges from 1 to $9 \times M$. Therefore, the number of parameter combinations is bounded by $M(9M)^3$, and it takes $O(M \times (9M)^3)$ time and space to build the memoization with top-down dynamic programming. Since $\mathbf{A}, \mathbf{B} \leq 10^{12}$ from the problem statement, we can derive that $M = 13$ and the number of parameter sets is at most $13 \times (9 \times 13)^3$, which is equal to $20,820,969$. Notice that this memoization can be built only once and reused in all test cases. Clearing and rebuilding the memoization for every test case may make the implementation time out since there are 100 test cases in test set 2.

Afterward, we can compute the value of $CountInterestingIntegers(N)$ by enumerating all possible values of S_{target} and recursively visiting each digit of N from left (0, the most significant digit) to right ($L_N - 1$, the least significant digit) to construct integers smaller than N and check the number of interesting integers with function f_2. The following is the pseudo code for $CountInterestingIntegers$:

```
def CountInterestingIntegers(N):
  if N == 0:
    return 0

  count = 0
  for S_target in (1 to (9 * CountDigits(N))):
    for L in (1 to (CountDigits(N) - 1)):
      count += CountInterestingIntegersWithNumberOfDigits(S_target, L)

    count += CountInterestingIntegersWithPrefixOfN(S_target, N, P=1, S=0,
                                                   digit_index=0, is_first_digit=True)
  return count

def CountInterestingIntegersWithNumberOfDigits(S_target, L):
  count = 0
  for digit in (1 to 9):
    count += f2(L - 1, P=(digit % S_target), S=digit, S_target)
  return count

def CountInterestingIntegersWithPrefixOfN(S_target, N, P, S, digit_index, is_first_digit):
  if digit_index == CountDigits(N):
    return 1 if P % S == 0 and S == S_target else 0

  if is_first_digit:
    digit_start = 1
  else:
    digit_start = 0

  count = 0
  for digit in (digit_start to (GetIthDigit(N, digit_index) - 1)):
    count += f2(CountDigits(N) - digit_index - 1, (P * digit) % S_target, S + digit, S_target)

  count += CountInterestingIntegersWithPrefixOfN(S_target, N,
                                                 (P * GetIthDigit(N, digit_index)) % S_target,
                                                 S + GetIthDigit(N, digit_index),
                                                 digit_index + 1, is_first_digit=False)
  return count
```

The overall time complexity is $O(M \times (9M)^3)$ to pre-compute the memoization of f_2, and $O(9M \times M)$ to get the number of interesting integers between $\mathbf{A}$ and $\mathbf{B}$ with memoization. Although method 2 adds one more parameter that is seemingly redundant, it has better and clearer time and space complexity estimates for building the memoization considering that the product can be a seemingly arbitrary number, while product modulo sum is bounded by a much smaller value. However, after solving all the test cases in test set 2, $2,406,887$ entries are kept in the memoization of f_2, which takes more time and space than method 1. We may apply some optimizations to f_2 to further reduce the memoization size and make it more performant, like returning 0 early when $S_{target} > 9 \times L$.