Factorial Trailing Digits

Author

Novemer 26, 2025

Contents

1 Formulating Problem

Problem: Find the last five digits before the trailing zeroes in $N!$

Let (N_n, b_d) where $N_n \coloneqq 10^n$, $b_d \coloneqq 10^d$, $b_d < N_n$ and $n, d \in \mathbb{N}^+$ be the problem of finding the last d digits of before the trailing zeros in $N!$. Then the problem we are interested in is (N_{12}, b_5).

Let $z(n)$ be a function that gives the number of trailing zeros in n. The $z(n!)$ is given by De Polignac's formula:

$$z(n!) = \sum_{i=1}^{\lfloor log_5(n) \rfloor} \lfloor \frac{n}{5^i} \rfloor \tag{1}$$

Using De Polignac's formula we can remove the trailing zeros to get the digits of interest.

$$(N_n, b_d) = \frac{N!}{2^{z(N!)} 5^{z(N!)}} \mod b \tag{2}$$

2 Grid Construction and Introduction to Series

For the problem (N_n, b_d) define $\lambda \coloneqq \frac{N}{b} = 10^{n-d}$. Then there is a grid G that can be constructed with dimensions is $b \times \lambda$.

$$\begin{array}{ccccccccc} 1 & 2 & \cdots & 5 & \cdots & 10 & \cdots & b \\ b+1 & b+2 & \cdots & b+5 & \cdots & b+10 & \cdots & b+b \\ 2b+1 & 2b+2 & \cdots & 2b+5 & \cdots & 2b+10 & \cdots & 2b+b \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ (\lambda-1)b+1 & (\lambda-1)b+2 & \cdots & (\lambda-1)b+5 & \cdots & (\lambda-1)b+10 & \cdots & (\lambda-1)b+b \end{array} \tag{3}$$

The product of all the elements of G is $N!$. Let c_v be the vth column of G. Then we define $c_v(i)$ as the ith element in column c_v.

$$c_v(i) = (i-1)b + v \quad i \in \{1 \cdots \lambda\} \tag{4}$$

The series c_v is an arithmetic sequence which can be seen by the following, $a_1 = v$ and $a_i = a_{i-1} + b$. Then let p_v be the product of the terms in c_v series.

$$p_v = \prod_{i=1}^{\lambda} (i-1)b + v \tag{5}$$

Then we have the series $p_1, p_2, \cdots p_b$ whose product is $N!$.

2.1 Building Solution from Column's of G

Given the series p_v from grid G built for problem (N_n, b_d) we can apply sequence lemma LS5 for $v|5$ else apply LS3. This will give the $\frac{N!}{5^{z(N!)}} \mod b$. Then to get the solution (N_n, b_d) we need to use sequence lemma LS4 to remove 2's for each 5 we removed. From LS4 we can remove λ 2's at a time. However, then we would need to multiple back the extra 2's that were taken off. This is fine and can be done mod b using lemma L1.

$$Z = \lceil \frac{z(N!)}{\lambda} \rceil \tag{6}$$

$$R = Z - z(N!) \tag{7}$$

There are $4\frac{b}{10}$ p_v that can have LS4 applied to them. The following construction hold if $Z \leq 4\frac{b}{10}$ else a construction of a similar form will need to be made which is fairly trivial with the machinery built up in the lemmas. The construction is as follows:

$$S_5 = \{5, 10, \cdots, b\} \tag{8}$$

$$S_2 \subset \{2, 4, \cdots, b\} \setminus S_5 \tag{9}$$

$$|S_2| = Z \tag{10}$$

$$S = \{1, 2, \cdots, b\} \setminus S_2 \cup S_5 \tag{11}$$

Let the $P_n(a_i) \mod b$ be the product of the first n terms of the series a_i mod b (defined by LS2). Let the $P_n^5(a_i) \mod b$ be the product of the first n terms of the series a_i dropping all 5's mod b (defined by LS3).

The construction takes out λZ 2's which is R more than the number of 2's need to be taken out which is $z(N!)$. So using lemma L1 we can add back 2^R.

$$(N_n, b_d) = \prod_{\forall v_i \in S, v_j \in S_2, v_k \in S_5} [P_\lambda(c_{v_i}) \mod b] \cdot [P_\lambda(\frac{1}{2} c_{v_j}) \mod b] \cdot [P_\lambda^5(c_k) \mod b] \cdot 2^R \pmod b \tag{12}$$

```python
def solve(N : int, b : int):
    s = time()
    assert(is_natural_number(N))
    assert(is_natural_number(b))
    assert(isValid_problem(N, b))
    N, b = int(N), int(b)

    z = number_of_trailing_zeros_of_factorial(N)
    l = N // b
    Z = math.ceil(z / l)
    R = Z*l - z
    assert(Z <= 4 * b/10)
    print(f'({N}, {b}) ::<>:: lambda = {l}; z(N!) = {z}; Z = {Z} R = {R}')

    product = 1
    for v in range(1, b + 1):
        if v % 5 == 0:
            # pass
            product *= product_of_sequence_drop_five(b, v, b, l)
        elif v % 2 == 0 and 0 < Z:
            product *= product_of_sequence(b//2, v//2, b, l)
            Z -= 1
        else:
            product *= product_of_sequence(b, v, b, l)

        assert(not is_multiple_of_five(product))
        product %= b

    product *= mod_pow(2, R, b)
    product %= b
    e = time()
    print(f'({N}, {b}) = {product} \t took {e-s} [s]')
```

3 Sequence

3.1 Lemma L1

Let $a, b \in \mathbb{N}$ then we can write a, b and ab with their 2's and 5's factored out:

$a = \alpha \cdot 2^{\alpha_2} \cdot 5^{\alpha_5}$
$b = \beta \cdot 2^{\beta_2} \cdot 5^{\beta_5}$
$ab = \lambda \cdot 2^{\gamma} \cdot 5^{\gamma}$ where $\gamma \coloneqq \min(\alpha_2 + \beta_2, \alpha_5 + \beta_5)$

The following two relations hold for any m.

$$\alpha\beta \cdot 2^{\alpha_2+\beta_2-\gamma} \cdot 5^{\alpha_5+\beta_5-\gamma} \equiv \lambda \pmod{m} \tag{13}$$

This then allow for the following:

$$(\frac{a}{2^\gamma 5^\gamma} \mod m)(\frac{b}{2^\gamma 5^\gamma} \mod m) \equiv \lambda \pmod{m} \tag{14}$$

3.2 Lemma LS1; Sequence is Purely Periodic

Given a sequence $a_i = (i-1) \cdot r + v$, $a_i \mod m$ where $i, r, v, m \in \mathbb{N}$ is purely periodic with period $\tau = \frac{m}{gcd(r,m)}$. First redefine a_i as $a_0 = v$ and $a_i = a_{i-1} + r$. Then we have $a_{k+l} = a_k + lr$ from the arithmetic progression.

$$a_{k+l} \equiv a_k \pmod{m} \leftrightarrow lr \equiv 0 \pmod{m} \leftrightarrow m|lr \tag{15}$$

Let the smallest m that satisfies $m|lr$ be x. Then $m|xr$ which implies that $m = \gcd(m, xr)$. One possibility of $x = m$, is there smaller number that satisfies the condition? If xr have a common divisor $d = gcd(x, r)$ then we get $x^* r^* d^2$ sill divides m by definition. Since $m|d$ hold always we can pull d out of out $x = m$ we can reduce x to $\frac{m}{gcd(r,m)}$ which is less than or equal to m and since there is no other larger number that we can pull out of both x^* and r^*, x is the smallest it can be then i.e. it is our period τ. [1]

3.3 Lemma LS2; Product of Periodic Sequence

Given a periodic sequence a_i with period τ and the product of the first n terms be p_n, then following allows for the computation of higher large n.

$$\prod_{i=1}^{n} a_i = p_{\tau}^{\lfloor \frac{\lambda}{\tau} \rfloor} \cdot p_{n \bmod \tau} \tag{16}$$

```python
# series a_i = (i - 1) * r + v
# calculates the the product of (a_1 ... a_n) mod b
def product_of_sequence(r : int, v : int, b : int, n : int) -> int:
    assert(isValid_sequence(r, v))
    assert(is_natural_number(b))
    assert(is_natural_number(n))
    # period of sequence mod b
    t = b // math.gcd(r, b)
    pt, pr = 1, 1
    for i in range(1, min(t, n) + 1):
        a_i = (i - 1) * r + v
        pt *= a_i % b
        pt %= b
        if i == n % t:
            pr = pt
    product = mod_pow(pt, n//min(n, t), b)
    if not n < t:
        product * pr
    return product % b
```

3.3.1 Series: (i-1)b + v mod b

We have the sequence $c_v(i) = (i-1)10^d + v = (i-1)2^d5^d + v$. We will use lemma L1, which requires $gcd(c_v, 5) = 1$. The only way $gcd(c_v, 5) > 1$ is if $gcd(v, 5) > 1$. So for the following we will deal with cases in which $gcd(v, 5) = 1$ that is to say v cannot be a multiple of 5. Since all our elements in our sequence not divisible by 5 lemma L1 can be used to get $\mod b$ product. Since our sequence is arithmetic we can use sequence lemma LS1 to get our sequence $\mod b$ has period $1 = \frac{b}{gcd(b,b)}$ and $c_v \mod b = v$. Using sequence lemma LS2 since we have a periodic sequence we get the following:

$$\prod_{i=1}^{n} a_i \equiv (p_\tau \mod b)^{\lfloor \frac{\lambda}{\tau} \rfloor} \cdot (p_{n \bmod \tau} \mod b) \pmod b \tag{17}$$

$$p_v \equiv v^\lambda \pmod b \tag{18}$$

3.3.2 Series: [(i-1)b/2 + v]/2 mod b

For the sequence $c_v(i) = (i-1)10^d + v = (i-1)2^d5^d + v$ we want the mod b of the product of $\frac{1}{2}c_v(i)$ first λ terms. Since $v|2$ and $gcd(v, 5) = 1$ we can use L1, LS1 and LS2.

$$\frac{1}{2}c_v(i) = (i-1)2^{d-1}5^d + \frac{v}{2} \tag{19}$$

Using LS1 we get the period of the new sequence as $\frac{2^d5^d}{gcd(2^{d-1}5^d, 2^d5^d)} = 2$. Then using LS2 we get the following and L1.

$$p_v \equiv [\frac{v}{2} \cdot (2^{d-1}5^d + \frac{v}{2}) \mod b]^{\lfloor \frac{\lambda}{2} \rfloor} \cdot (\frac{v}{2})^{\lambda \bmod b} \pmod b \tag{20}$$

3.4 Lemma LS3: Product of Series dropping 5's mod b

Let the series be of the form $a_i = (i-1)\cdot r + v$. From sequence lemma LS1 we know that a_i is purely periodic with period $\tau = \frac{b}{gcd(r,b)}$. Find the following product:

$$\prod_{i=1}^{\lambda} \frac{a_i}{5^{p_5^i}} \pmod b \tag{21}$$

where p_5^i is the exponent for the factor 5 for a_i's factorization. Let $r = r^* 5^{p_r}$ and $v = v * 5^{p_v}$ where $gcd(r^*, 5) = gcd(v^*, 5) = 1$. Then the series $a_i^* = (i-1)\cdot r^* + v^*$ product of first λ terms dropping 5's mod b is the same as our original problem.

case: $p_v < p_r$

The series will not have any multiple of 5's i.e. $gcd(a_i^*, 5) = 1 \forall i \in \mathbb{N}$. Therefore sequence lemma LS2 can be used to find the product mod b.

case: $p_r \leq p_v$

The series will of the following form $a_i^* = (i-1)\cdot r^* + v^* 5^{p_v - p_r}$. When does $a_i^* | 5$? The second term will have end in $\{5\}$ when $p_r < p_v$ and will be $\{1,2,3,4,6,7,8,9\}$ when $p_r = p_v$, it could end in 0 iff $v = 0$. Without specifics analysis to find to smallest i for $a_i^* | 5$ must be done manually. To do this take the array $[0, \cdots, 9](r^* \mod 10) + (v^* \mod 10)$ and find the first instance of a multiple of 5, the index (1-indexed) of this position be O_5. Then $a_{O_5}^* | 5$ is the first element in the series that is divisible by 5. The sequence $a_{O_5}^*, a_{O_5+5}^*, \cdots$ are all the terms that are multiple of 5.

Therefore, we can use sequence lemma LS2 for the terms in a_i^* where $i \notin \{O_5, O_5 + 5, \cdots\}$ iff $O_5 \leq 5 \wedge \tau | 5$. The condition on the period of the sequence is from the fact that each period of the sequence needs to take out each instance of $a_i^* | 5$ and need to line up with with in the period; this is true if the period of $a_i^* | 5$ 5 is a factor of τ and $i \neq O_5 + 5k \ \forall k \in \mathbb{Z}$.

Therefore here does not seem to be a pattern for the terms $a_{O_5+5k}^*$ this can be seen by $a_{O_5+5k}^* | 5^2$ is periodic but it will have its own O_5^2 that needs to be calculated. There will be at most $n_O = \lfloor \log_5 \lambda \rfloor$ offsets that will needed to be calculated. And each offset $O_{5^{p_O}}$ the following must be true:

$$\tau | 5^{p_O} \tag{22}$$
$$O_{5^{p_O}} \leq \tau \tag{23}$$
$$O_{5^{p_O}} \leq 5^{p_O} \tag{24}$$
$$5^{p_O} \leq \tau \tag{25}$$
$$\tag{26}$$

These condition are not generally going to be hold for $1 < p_O$, so the series $a_{O_5+5k}^*$ will need to be brute forced.

```python
# series a_i = (i - 1) * r + v
# calculates the the product of (a_1 ... a_n) / 5^p mod b
# where p is the number of factors in the product
def product_of_sequence_drop_five(r : int, v : int, b : int, n : int) -> int:
    assert(isValid_sequence(r, v))
    assert(is_natural_number(b))
    assert(is_natural_number(n))
    assert(is_multiple_of_five(b))
    # remove trivial factors of 5 in the series
    pr = factors_of_five(r)
    pv = factors_of_five(v)
    min_factors_of_five = min(pr, pv)
    reduce_sequence_by = 5**min_factors_of_five
    r = r // reduce_sequence_by
    v = v // reduce_sequence_by
    assert(not is_multiple_of_five(math.gcd(r, v)))

    if pv < pr:
        return product_of_sequence(r, v, b, n)

    # period of the sequence
    t = b // math.gcd(r, b)

    # check if series has a any more multiple of 5's
    offset = first_multiple_of_five(r, v)
    assert(offset <= 5)
    # ignore the a_i|5
    pt, pr = 1, 1
    for i in range(1, min(t, n) + 1):
        if offset is not None and is_multiple_of_five(i - offset):
            continue
        a_i = (i - 1) * r + v
        assert(not is_multiple_of_five(a_i))
        pt *= a_i % b
        pt %= b
        if i == n % t:
            pr = pt
    ignore_ever_fifth = mod_pow(pt, n//min(n, t), b)
    if not n < t:
        ignore_ever_fifth = (ignore_ever_fifth * pr) % b

    # get product of a_i that where skipped
    p5s = 1
    if offset is not None:
        for i in range(offset, n + 1, 5):
            a_i = drop_fives((i - 1) * r + v)
            p5s *= a_i % b
            p5s %= b
    return (ignore_ever_fifth * p5s) % b
```

4 Helper Methods

```python
from functools import lru_cache
import math
from time import time

@lru_cache
def is_whole_number(n : int) -> bool:
    if isinstance(n, int):
        return True
    elif isinstance(n, float):
        return n.is_integer()
    else:
        return False

@lru_cache
def is_natural_number(n) -> bool:
    return not n == 0 and is_whole_number(n)

@lru_cache
def is_multiple_of_five(n : int) -> bool:
    return is_whole_number(n) and (n % 10 in {0, 5})

# number of trailing zeros for n!
def number_of_trailing_zeros_of_factorial(n : int) -> int:
    assert(is_natural_number(n))
    n = int(n)
    R = range(1, int(math.log(n) / math.log(5)) + 1)
    z = lambda i : n // 5**i
    return sum(map(z,  R))
assert(number_of_trailing_zeros_of_factorial(9) == 1)
assert(number_of_trailing_zeros_of_factorial(10) == 2)
assert(number_of_trailing_zeros_of_factorial(20) == 4)

PRIMES = [2]

@lru_cache
def prime_factors(n : int) -> dict:
    assert(is_natural_number(n))
    global PRIMES
    n = int(n)
    f : dict[int][int] = {}

    last_prime = None
    for p in PRIMES:
        while n % p == 0:
            n //= p
            f[p] = f.get(p, 0) + 1
        last_prime = p
        if n < p*p:
            break
    assert(last_prime is not None)
    q = last_prime
    while q*q <= n:
        if n % q == 0:
            PRIMES.append(q)
        while n % q == 0:
            n //= q
```

```
            f[q] = f.get(q, 0) + 1
        q += 1
    if n > 1:
        PRIMES.append(n)
        f[n] = f.get(n, 0) + 1
    return f
assert(prime_factors(2**2 * 5**33 * 7) == {2:2, 5:33, 7:1})

@lru_cache
def factors_of_five(n):
    if n == 0:
        return 0
    p = 0
    while n % 5 == 0:
        n //= 5
        p += 1
    return p

# computes a^n mod b
# implements Right-to-left binary method
# [https://en.wikipedia.org/wiki/Modular_exponentiation#Right-to-left_binary_method]
@lru_cache
def mod_pow(a : int, n : int, b : int) -> int:
    assert(is_natural_number(a))
    assert(is_natural_number(b))
    assert(is_whole_number(n) and 0 <= n)
    if b == 1:
        return 0
    # (b-1) * (b-1) should not overflow
    product = 1
    a = a % b
    while 0 < n:
        if n % 2 == 1:
            product = (product * a) % b
        n = n >> 1
        a = (a * a) % b
    return product

def isValid_b(b : int) -> bool:
    return math.log10(b) % 1 == 0 and b > 1

def isValid_N(N : int) -> bool:
    return math.log10(N) % 1 == 0 and N > 1

def isValid_problem(N : int, b: int) -> bool:
    return isValid_N(N) and isValid_b(b) and b <= N

def isValid_sequence(r : int, v : int) -> bool:
    return r % 1 == 0 and is_natural_number(v) and 0 < v

def drop_fives(n : int) -> int:
    if n == 0:
        return n
    while n % 5 == 0:
        n //= 5
    return n
assert(drop_fives(5**23 * 7) == 7)

```

```
def first_multiple_of_five(r : int, v : int) -> int | None:
    assert(is_natural_number(r))
    assert(is_natural_number(v))
    for d in range(10):
       if is_multiple_of_five((d*r + v) % 10):
           return d + 1
    return None
```