Kick Start 2017 - Round G

Analysis: Huge Numbers

Huge Numbers : Analysis

Small dataset

For the small input, calculating the actual value of **N** factorial suffices, since **N** is up to 10. We just need to calculate $\mathbf{A}^n$ mod **P**, where n could be up to 10! = 3628800. We can compute this iteratively, maintaining our answer modulo **P** at all times, as in the following pseudocode:

```
ans = 1
for i = 1 to factorial(N)
  // Since, multiplication is associative modulo P,
  // we can maintain our answer modulo P
  ans = (ans * A) % P
return ans
```

Since **P** and **A** are both no greater than 10^5, and we are taking modulo **P** at each stage, we do not need to worry that (ans * **A**) will overflow the result, provided that we use a long rather than an int to store ans.

Large dataset

At first, it may seem like this problem requires a number-theoretic approach. But there exists a very simple solution which employs fast exponentiation.

First, let's see how efficiently we can calculate $\mathbf{A}^n$ mod **P** for a given n. We can use a divide and conquer approach to come up with an O(log n) solution, as summarized by the following algorithm:

```
pow(a, n, p):
  if n == 0
    return 1

  pow_half = pow(a, n / 2, p)
  pow_half_sq = (pow_half  * pow_half) % p // again, multiplication is associative modulo p
  if n % 2 == 0
    return pow_half_sq
  else
    return (pow_half_sq * a) % p
```

We can also take advantage of a basic property of exponents: a^{b*c} can be rewritten as a^{b^c}. So, we can write $\mathbf{A}^{\mathbf{N!}}$ as $\mathbf{A}^{1^{2^{3\ldots}}}$. And, since multiplication modulo **P** is associative, we can maintain our answer modulo **P** at all times. So, our O(**N** log **N**) algorithm is:

```
ans = A % P
for i = 2 to N
  ans = pow(ans, i, P)
return ans
```