QPOLYSUM – Editorial
CodeChef December 2012 Long Contest

PRELIMINARIES

Modular Multiplication

In this problem, M does not fit in 32-bit integer type. Therefore, the value of $A \times B \bmod M$ for $0 \le A, B \le M$ cannot be calculated by direct multiplication in 64-bit integer type. One of the ways to do this fast enough is to use multiplication by doubling (i.e., the summation equivalent for exponentiation by squaring).

First, we represent A in binary: $A = a_{k-1} 2^{k-1} + a_{k-2} 2^{k-2} + \ldots + a_0 2^0$, where a_i is either 0 or 1. Then, $A \times B = a_0 \times B + a_1 \times 2B + a_2 \times 4B + \ldots + a_{k-1} \times 2^{k-1} B$. Each term in the sequence B, $2B$, $4B$, … can be calculated by doubling the previous term. Since the only operations are addition and doubling, all intermediate results will never exceed $2M$ and thus can be calculated in 64-bit integer type. For example, suppose we want to calculate 13×7. Since 13 = 1101 in binary, then $13 \times 7 = 7 + 28 + 56 = 91$.

Here is a pseudocode of this algorithm. This algorithm will run in $O(\log M)$ time and uses very simple math operations. See setter's solution for efficient implementation.

```
function multiply(A, B):
    res = 1
    while A > 0:
        if (A mod 2) == 1:
            res = (res + B) mod M
        B = (2 × B) mod M
     return res
```

Modular Inverse

In this solution, we often need to calculate modular inverses modulo M. Since M is not prime in general, we cannot use Fermat's little theorem $x^{-1} \equiv x^{M-2} \pmod M$. We should use the more general **extended Euclid's algorithm** instead that runs in $O(\log M)$. Moreover, this will ensure that all intermediate results will not exceed M by absolute value. Therefore, we can calculate modular inverse in 64-bit integer type.

In this editorial, we will use the notation x^{-1} as the modular inverse of x modulo M.

Binomial Coefficient

In this solution, we need to be able to calculate the sequence $\left(C_n^j \bmod M\right)_{j=0}^{d}$, where C_n^j is the number of ways to choose j objects from n objects **and** the least prime factor of M is greater than d. We can use the recurrence formula $C_n^j = C_n^{j-1}(n-j+1)/j$. We can use modular inverses here because M and j are always coprime. So, $C_n^j \bmod M = C_n^{j-1}(n-j+1)\, j^{-1} \bmod M$. Moreover, since there are only multiplication operations, we can take n modulo M as well. Therefore, $C_n^j = C_{n \bmod M}^j$ in all parts of this solution.

This sequence can be calculated in $O(d \log M)$ time. Note that we need $O(\log M)$ time for calculating the modular inverse of j as well as $O(\log M)$ operations to multiply three integers C_n^{j-1}, $n - j + 1$, and j^{-1}.

Calculating N mod M

In this solution, we will need to calculate $N \bmod M$ in order to calculate $C_N^j = C_{N \bmod M}^j$. However, N is very large in this problem. We can calculate it using a method that is similar to **Horner's method**. For example,

$$123 \bmod M = ((((1 \times 10) + 2) \times 10) + 3) \bmod M$$

and perform all operations modulo M.

Calculating Q^N mod M

In this solution, we will need to calculate $Q^N \bmod M$. We can calculate it using a method that is similar to Horner's method as well. For example,

$$Q\text{^}123 = Q\text{^}((((1 \times 10) + 2) \times 10) + 3) \bmod M = ((((Q^1)\text{^}10 \times Q^2)\text{^}10) \times Q^3) \bmod M$$

Here is a pseudocode of this algorithm. Here $N[i]$ denotes the i-th least significant digit of N (0-based). We will also need to calculate the 10th power of an integer in a fast way as denoted in this algorithm.

```
function powLarge(Q, N):
    powQ[0] = 1
    for i = 1; i < 10; i++:
        powQ[i] = (Q × powQ[i-1]) mod M
    res = 1
    for i = 0; i < length(N); i++:
        temp = res
        res = multiply(res, res)        // res = temp^2
        res = multiply(res, res)        // res = temp^4
        res = multiply(res, temp)       // res = temp^5
        res = multiply(res, res)        // res = temp^10
        res = multiply(res, powQ[N[i]]) // res = temp^10 × Q^N[i]
    return res
```

The complexity of this algorithm is $O(\log N \log M)$.

Convolution

In this solution, several times we need to efficiently calculate the convolution of two integer sequences modulo M. Let $x = (x_0, x_1, \ldots, x_d)$ and $y = (y_0, y_1, \ldots, y_d)$ be two sequences. Their convolution $z = (z_0, z_1, \ldots, z_d)$ is defined as $z_k = x_0y_k + x_1y_{k-1} + \ldots + x_ky_0$. The convolution of two sequences of length d can be calculated in $O(d \log d)$ time using fast Fourier transform (FFT). In this problem, all calculations must be done modulo M, so we will use **number-theoretic transform** (FFT performed in modulo prime) instead.

However, M is too large and is not always prime in this problem. To overcome this, we will calculate the convolution of x and y modulo K primes $P_1, P_2, \ldots, P_K$ that fit in 32-bit integer type such that their product $P = P_1 P_2 \ldots P_K$ is larger than M^2D, the upper bound of the maximum value in the convolution. Since $z_k < M^2D < P$, then $z_k \bmod P = z_k \bmod M$ for all k. The value of $z_k \bmod P$ can be calculated using **Chinese remainder theorem** (CRT) algorithm. Finally, the value of $z_k \bmod M$ can be restored by calculating $(z_k \bmod P) \bmod M$.

To calculate convolutions using number-theoretic transform, each prime must have the form $P_i = n\,k + 1$ where n is a power of two and is larger than $2D$, the maximum length of the convolution result. We need to find the integer equivalent of the principal primitive n-th root of unity ω_n, where $\omega_n^n = 1$ in the usual FFT algorithm. Here is how to calculate ω_n. Let r be a primitive root of P_i. Then, ω_n = the principal n-th root of unity = $r^k \bmod M$. The rest of the number-theoretic transform is similar to FFT but is performed in integers modulo P_i.

The primes and the corresponding primitive roots can be found beforehand using any math package such as **Mathematica** or **Maple**. Under the constraints of the problem, we need 5 such primes in the worst case. Then, we restore the actual convolution using CRT and then take the remainder modulo M. The author's solution uses **Garner's algorithm** to perform the CRT so that we do not need to construct the convolution modulo P using arbitrary-precision arithmetics.

Let $K = \log(M^2 d)/\log(2^{31})$ be the approximate number of primes we need to calculate the convolution. The overall complexity to calculate convolution modulo M is $O\left(K\,d \log d + d\left(K^2 + \log(M^2 d)\right)\right)$. Here, $Kd \log d$ corresponds to K convolutions modulo primes that fit in 32-bit integer type, K^2 is the complexity of Garner's algorithm, and $\log M^2D$ is the complexity needed to restore the answer modulo M considering the **Modular Multiplication** section.

Polynomial

Every polynomial $P(x)$ of degree at most d can be uniquely represented as

$$P(x) = \sum_{k=0}^{d} a_k C_x^k$$

where $C_x^k = \frac{x!}{k!(x-k)!} = \frac{x\cdot(x-1)\cdot\ldots\cdot(x-k+1)}{1\cdot 2\cdot\ldots\cdot k}$. It can be proved that $a_k = \sum_{j=0}^{k} (-1)^{k-j} C_k^j P(j)$.

In the solution below we several times will need to find the sequence $a_k \bmod M$ by the values $P(0)$, $P(1)$, …, $P(D)$ modulo M, where M is coprime to all numbers up to d. For this let's consider the sequence $\tilde{a} = (a_k/k!)_{k=0}^{d}$. Note that

$$\frac{a_k}{k!}=\sum_{j=0}^{k}(-1)^{k-j}\frac{1}{j!(k-j)!}P(j)=\sum_{j=0}^{k}\frac{(-1)^{k-j}}{(k-j)!}\times\frac{P(j)}{j!}.$$

Hence $\tilde{a}$ is the convolution of sequences $b=\left((-1)^k/k!\right)_{k=0}^{d}$ and $c=\left(P(k)/k!\right)_{k=0}^{d}$. Since M is coprime to all numbers up to d we can calculate b and c using modular inverses and then find $\tilde{a}$ using the convolution method described before. Thus, given $P(0)$, $P(1)$, …, $P(d)$ mod M, we can calculate values $a_0/0!$, $a_1/1!$, …, $a_d/d!$ mod M quite efficiently. If we multiply the k-th term by $k!$, we can obtain the values a_0, a_1, …, a_d; i.e., the coefficients of the polynomial in the above representation.

ACTUAL SOLUTION

First let's handle a special case when $Q = 0$. Clearly, the answer is just $P(0)$. Otherwise, let's split M into a product of two integers:

$$M = M_1 M_2$$

where $\gcd(Q - 1, M_1) = 1$ and $(Q - 1)^u$ divides M_2 for some integer u. If we know factorization of M, this can be calculated easily by moving all common prime factors of M and Q to M_1 and all other prime factors of M to M_2. Fortunately, this can be done without factorization using the following simple loop:

```
function split(M, Q):
    M1 = M
    while true:
        g = gcd(M1, Q - 1)
        if g == 1 then break
        M1 /= g
    return (M1, M / M1)
```

In each step in the while-loop, we remove from M_1 a portion of common prime factors of M and $Q - 1$. This loop will terminate when all prime factors have been transferred to M_2. In the end, M_1 and M_2 will satisfy the required properties.

We will calculate the answer modulo M_1 and modulo M_2 will separately and restore the final answer using Chinese remainder theorem.

Let's consider both cases.

Calculating Answer Modulo M_1

For convenience, let $M = M_1$ in this section. Note that M and $Q - 1$ are coprime so that the modular inverse $(Q - 1)^{-1}$ mod M exists. Then, we can use the following neat formula:

$$\sum_{k=0}^{N-1}C_k^j Q^k=Q^j(1-Q)^{-j-1}\left(1-Q^N\sum_{k=0}^{j}C_N^k Q^{j-k}(1-Q)^k\right)$$

which can be proved by induction on j using differentiation on Q.

Let $P(k)=\sum_{j=0}^{D}a_j C_k^j$. Numbers a_0, a_1, …, a_D can be calculated using the method from **Polynomial** section in $O\left(K\,D\log D+D\left(K^2+\log\left(M^2 D\right)\right)\right)$ time. Now, we have

$$\sum_{k=0}^{N-1}P(k)Q^k=\sum_{k=0}^{N-1}\sum_{j=0}^{D}a_j C_k^j Q^k=\sum_{j=0}^{D}a_j\sum_{k=0}^{N-1}C_k^j Q^k=\sum_{j=0}^{D}a_j Q^j(1-Q)^{-j-1}-Q^N\sum_{j=0}^{D}\sum_{k=0}^{j}a_j C_N^k Q^{j-k}(1-Q)^{k-j-1}.$$

Let $b_k=C_N^k$ and $c_k=Q^k(1-Q)^{-k-1}$. The double sum in the right-hand side of the last equality can be rewritten as:

$$\sum_{j=0}^{D}\sum_{k=0}^{j}a_j C_N^k Q^{j-k}(1-Q)^{k-j-1}=\sum_{j=0}^{D}\sum_{k=0}^{j}a_j b_k c_{j-k}=\sum_{j=0}^{D}a_j\sum_{k=0}^{j}b_k c_{j-k}=\sum_{j=0}^{D}a_j y_j$$

where $y = (y_0, y_1, \ldots, y_D)$ is a convolution of sequences b and c. The sequence b can be calculated in $O(D \log M)$ time using method from **Binomial Coefficient** section. The sequence c can be calculated in $O(D \log M)$ time as well. Here, we spend additional $O(\log M)$ time to calculate $(1 - Q)^{-1}$ mod M before calculating the sequence c. Then, y can be calculated in $O\left(K\,D\log D+D\left(K^2+\log\left(M^2 D\right)\right)\right)$ time using the method described

Convolution section. Q^N mod M can be calculated in $O(\log N \log M)$ time using the method described in **Calculating Q^N mod M** section.

Finally, the answer can be calculated in $O(D \log M)$ time as $\sum_{j=0}^{D} a_j \left(c_j - Q^N y_j\right)$.

Calculating Answer Modulo M_2

Recall that M_2 divides $(Q-1)^u$ but does not divide $(Q-1)^{u-1}$. Let $M_2 = p_1^{a_1} \cdot \ldots \cdot p_k^{a_k}$ be a prime factorization of M_2. Clearly, $u \le \max(a_1, \ldots, a_k)$. Since M is not divisible by any number from 2 to D+14, it follows that minimal prime factor of M is 17. Further, note that $17^{15} > 10^{18}$, hence $a_i \le 14$ for all i. And so $u \le 14$. Therefore, ***M* is not divisible by any number from 2 to *D*+*u*.** This will be crucial below.

For convenience, let $M = M_2$ in this section from now. By the **Binomial theorem**, the following formula holds for any non-negative integer k:

$$Q^k = \left(1 + (Q-1)\right)^k = \sum_{j=0}^{k} C_k^j (Q-1)^j \equiv \sum_{j=0}^{u-1} C_k^j (Q-1)^j \pmod{M}.$$

Clearly, the last term $\sum_{j=0}^{u-1} C_k^j (Q-1)^j$ is a polynomial of degree $u - 1$ in the representation as mentioned in the **Polynomial** section. Therefore, $T(k) = P(k)\, Q^k$ is a polynomial of degree at most $D + u - 1$. We will represent $T(k)$ in the representation mentioned in the **Polynomial** section as well. Therefore, we need the values of $T(0)$, $T(1)$, …, $T(D + u - 1)$. This can be calculated in a single loop as follows.

```
powQ = 1
for i = 0; i ≤ D+u-1; i++:
    T(0) = (P(i) × powQ) mod M
    powQ = (powQ × Q) mod M
```

However, we are only given $P(0)$, $P(1)$, …, $P(D)$ modulo M as the input. The remaining values $P(D + 1)$, $P(D + 2)$, … $P(D + u - 1)$ mod M can be calculated using recurrence relation $P(k) = \sum_{j=1}^{D+1} (-1)^{j-1} C_{D+1}^j P(k-j)$ in $O(D\, u \log M)$ time after calculating the sequence $\left(C_{D+1}^j \bmod M\right)_{j=0}^{D+1}$ in $O(D \log M)$ time using method described in **Binomial Coefficient** section. Here we used that M is not divisible by any number from 2 to $D + 1$. Note that $M_2^u \le M$, hence $O(D\, u \log M_2) = O(D \log M)$ actually.

Using method described in the **Polynomial** section, we can calculate the coefficient t_0, t_1, …, t_{D+u-1} such that $T(k) = \sum_{j=0}^{D+u-1} t_j C_k^j$ in $O\left(K\, D \log D + D\left(K^2 + \log\left(M^2 D\right)\right)\right)$ time. To apply this we need M to be not divisible by any number from 2 to D+u-1, and we really have this due to the bold sentence from the first paragraph of this case. Finally, by the identity for binomial coefficients, we obtain

$$\sum_{k=0}^{N-1} P(k) Q^k = \sum_{k=0}^{N-1} T(k) = \sum_{k=0}^{N-1} \sum_{j=0}^{D+u-1} t_j C_k^j = \sum_{j=0}^{D+u-1} t_j \sum_{k=0}^{N-1} C_k^j = \sum_{j=0}^{D+u-1} t_j C_N^{j+1}.$$

Therefore, by calculating the sequence $\left(C_n^j \bmod M\right)_{j=0}^{D+u}$ in $O(D \log M)$ time using the method described in **Binomial Coefficient** section, we get a fast enough algorithm in this case. And here according to **Binomial Coefficient** section we need M to be not divisible by any number from 2 to D+u. So the mentioned bold sentence play his role completely :)

Conclusion

The overall time complexity of the described solution is

$$O\left(D\left(K \log D + K^2 + \log\left(M^2 D\right)\right) + \log N \cdot \log M\right)$$

Here, $K \log D + K^2 + \log\left(M^2 d\right)$ is the complexity of the convolution. Many other $O(D \log M)$ parts of the solution are not included in the complexity since $\log M < \log (M^2 D)$. Of course, by D here we mean max(D, 1) since in the special case $D = 0$ all these formulas are useless. Finally, $\log N \log M$ corresponds to calculation of Q^N mod M.