GUÍA DE APRENDIZAJE PARA CONCURSANTES ICPC Y IOI: ARITMÉTICA MODULAR

1. Introducción

En varios problemas nos podemos encontrar que se nos pide que la solución final debe ser modulada con respecto a cierto valor. Un ejemplo de lo anterior nos lo podemos encontrar sobre todo en la especificación de salida de ciertos problemas como por ejemplo:

Una única línea con el número posible de secuencias que Octavia puede obtener, módulo 1000000007

En la siguiente guía vamos abordar como enfocar nuestra salida o proceso de solución cuando la solución debe ser modulada.

2. Conocimientos previos

2.1. Operador módulo

El operador módulo representado en la mayoría de los lenguajes de programación con símbolo (%) da como resultado el resto de la división entera. Por ejemplo 20 %7 da como resultado 6 que es el resto de la división entre 20 y 7.

2.2. El teorema del cociente y del residuo

Dado cualquier entero A y un entero positivo B, existen dos enteros únicos Q y R tales que: A= B * Q + R donde 0 $\leq$ R <B Podemos ver que esto viene directamente de la división larga. Cuando dividimos A entre B en la división larga, Q es el cociente y R es el residuo. Si podemos escribir un número en esta forma, entonces A mod B = R.

2.3. Función Totient de Euler

Función totient de Euler, también conocida como función $\phi(n)$, cuenta el número de enteros entre 1 y n inclusive, que son coprimos de n. Dos números son coprimos si su máximo común divisor es igual 1 (1 se considera coprimo de cualquier número).

2.4. División Euclidiana

En aritmética , la división euclidiana , o división con resto , es el proceso de dividir un número entero (el dividendo) por otro (el divisor), de manera que se produce un cociente entero y un resto de número natural estrictamente menor que el valor absoluto del número . divisor. Una propiedad fundamental es que el cociente y el resto existen y son únicos, bajo ciertas condiciones. Debido a esta singularidad, la división euclidiana a menudo se considera sin hacer referencia a ningún método de cálculo y sin calcular explícitamente el cociente y el resto. Los métodos de cálculo se denominanalgoritmos de división de enteros , el más conocido de los cuales es la división larga .

3. Desarrollo

En matemáticas, la aritmética modular (es una categoría especial de aritmética que utiliza solo enteros. En otras palabras, la aritmética modular es la aritmética de la congruencia. La aritmética modular a veces se conoce como aritmética de reloj, ya que uno de los usos más conocidos de la aritmética modular es el reloj de 12 horas, que tiene el período de tiempo dividido en dos mitades iguales.

En su libro *Disquistiones Arithmeticae* publicado en 1801, Carl Friedrich Gauss introdujo el enfoque moderno de la aritmética modular. Según las matemáticas, la aritmética modular se considera la aritmética de cualquier imagen homomórfica no trivial del anillo de enteros. En aritmética modular, los números que se tratan son solo enteros y las operaciones que se usan son solo suma, resta, multiplicación y división. En aritmética modular, los números se envuelven o redondean al alcanzar un cierto valor, haciendo uso del módulo. En esta forma de aritmética, se consideran los restos. La aritmética modular generalmente se asocia con números primos. Dos números se consideran equivalentes, el resto de ambos números dividido por un número único es igual.

3.1. Suma (a+b) % m

Probaremos que **(A + B) mod C = (A mod C + B mod C) mod C**

Debemos mostrar que LI=LD

A partir del teorema del cociente y del residuo podemos escribir A y B como:
A = C * Q1 + R1 donde 0 $\leq$ R1 <C y Q1 son enteros. A mod C = R1
B = C * Q2 + R2 donde 0 $\leq$ R2 <C y Q2 son enteros. B mod C = R2

(A + B) = C * (Q1 + Q2) + R1+R2
LI = (A + B) mod C
LI = (C * (Q1 + Q2) + R1+ R2) mod C

Podemos eliminar los múltiplos de C cuando tomamos mod C.
LI = (R1 + R2) mod C
LD = (A mod C + B mod C) mod C
LD = (R1 + R2) mod C
LI=LD= (R1 + R2) mod C

3.2. Multiplicación (a*b) % m

Probaremos que **(A * B) mod C = (A mod C * B mod C) mod C**

Debemos mostrar que LI = LD

A partir del teorema del cociente y del residuo podemos escribir A y B como:

A = C * C1 + R1 donde $0 \leq$ R1 <C y C1 es un integral. A módulo C = R1
B = C * C2 + R2 donde $0 \leq$ R2 <C y C2 es un integral. B módulo C = R2

LI = (A * B) mod C
LI = ((C * C1 + R1) * (C * C2 + R2)) mod C
LI = (C * C * C1 * C2 + C * C1 * R2 + C * C2 * R1 + R1 * R2) mod C
LI = (C * (C * C1 * C2 + C1 * R2 + C2 * R1) + R1 * R2) mod C

Podemos eliminar los múltiplos de C cuando tomamos el módulo C:
LI = (R1 * R2) mod C

Ahora hagamos el LD
LD = (A mod C * B mod C) mod C
LI = (R1 * R2) mod C

Por lo tanto, LD = LI
LI = LD = (R1 * R2) mod C

3.3. Resta (a-b) % m

La desmostración de la operación de la resta es similar a la suma pero con un detalle importante el $b \mod m$ puede ser mayor que el $a \mod m$ por lo que quedaría un valor negativo lo cual no sería correcto pero haciendo un ajuste de corrimiento de una vuelta completa del anillo que se genera con los posibles restos de $\mod m$ se resuelve el problema por lo que podemos concluir que :

(A - B) mod C = (A mod C - B mod C + C) mod C

Note que se le suma el valor de C para provocar la vuelta al anillo.

3.4. División (a/b) % m

La división la vamos a trabajar como una mulplicación con el siguiente proceso partiendo de lo siguiente:

$$\frac{a}{b} \mod m$$

vamos a plantear la división como una multiplicación de la siguiente manera:

$$(a \times \frac{1}{b}) \mod m$$

$$(a \times b^{-1}) \mod m$$

Aplicando lo conocido de la multiplicación nos quedaría de la siguiente manera:

$$(a \mod m \times b^{-1} \mod m) \mod m$$

Con el primer término no debe existir duda pero con el segundo término $b^{-1} \mod m$ nos queda ver como operar. Si estudiamos un poco de congruencia nos podemos percatar que podemos sustituir el término b^{-1} por su inverso multiplicativo al cual le vamos a dedicar un espacio a continuación.

3.4.1. Inverso Multiplicativo Modular

Un inverso multiplicativo modular de un entero a es un entero x tal que $a \cdot x$ es congruente con 1 modular algún módulo m. Para escribirlo de manera formal: queremos encontrar un número entero x de modo que

$$a \cdot x \equiv 1 \mod m.$$

También denotaremos x simplemente con a^{-1}.

Debemos tener en cuenta que el inverso modular no siempre existe. Por ejemplo, deja $m = 4$, $a = 2$. Comprobando todos los valores posibles módulo m debe quedar claro que no podemos encontrar a^{-1} satisfaciendo la ecuación anterior. Se puede probar que el inverso modular existe si y solo si a y m son relativamente primos (es decir $\gcd(a, m) = 1$).

Presentamos dos métodos para encontrar el inverso modular en caso de que exista, y un método para encontrar el inverso modular para todos los números en tiempo lineal.

3.4.2. Encontrar el inverso modular usando el algoritmo euclidiano extendido

Considere la siguiente ecuación (con incógnita x y y):

$$a \cdot x + m \cdot y = 1$$

Esta es una ecuación Diofántica Lineal en dos variables. Cuando $\gcd(a, m) = 1$, la ecuación tiene una solución que se puede encontrar utilizando el algoritmo euclidiano extendido . Tenga en cuenta que $\gcd(a, m) = 1$ es también la condición para que exista el inverso modular.

Ahora bien, si tomamos módulo m de ambos lados, podemos deshacernos de $m \cdot y$, y la ecuación se convierte en:

$$a \cdot x \equiv 1 \mod m$$

Así, el inverso modular de a es x.

3.4.3. Encontrar el inverso modular usando exponenciación binaria

Otro método para encontrar el inverso modular es usar el teorema de Euler, que establece que la siguiente congruencia es verdadera si a y m son relativamente primos:

$$a^{\phi(m)} \equiv 1 \mod m$$

ϕ es la función Totient de Euler . De nuevo, tenga en cuenta que a y m ser primo relativo también era la condición para que existiera el inverso modular.

Si m es un número primo, esto se simplifica al pequeño teorema de Fermat :

$$a^{m-1} \equiv 1 \mod m$$

Multiplica ambos lados de las ecuaciones anteriores por a^{-1} , y obtenemos:

- Para un módulo arbitrario (pero coprimo) m:$a^{\phi(m)-1} \equiv a^{-1} \mod m$
- Para un módulo primo m:$a^{m-2} \equiv a^{-1} \mod m$

A partir de estos resultados, podemos encontrar fácilmente el inverso modular usando el algoritmo de exponenciación binaria , que funciona en $O(\log m)$ tiempo.

Aunque este método es más fácil de entender que el método descrito en el párrafo anterior, en el caso de que m no es un número primo, necesitamos calcular la función phi de Euler, que implica la factorización de m, que puede ser muy difícil. Si la factorización prima de m se conoce, entonces la complejidad de este método es $O(\log m)$.

3.4.4. Encontrar el inverso modular usando la división euclidiana

Dado que $m > i$ (o podemos modular para hacerlo más pequeño en 1 paso), según la división euclidiana

$$m = k \cdot i + r$$

dónde $k = \left\lfloor \frac{m}{i} \right\rfloor$ y $r = m \mod i$, entonces

$$\begin{aligned}
\implies & \quad 0 \equiv k \cdot i + r && \mod m \\
\iff & \quad r \equiv -k \cdot i && \mod m \\
\iff & \quad r \cdot i^{-1} \equiv -k && \mod m \\
\iff & \quad i^{-1} \equiv -k \cdot r^{-1} && \mod m
\end{aligned}$$

Una vez visto las formas de encontrar el inverso multiplicativo modular solo nos queda sustituir donde está b^{-1} por su inverso multiplicativo al cual llamaremos B y de esta forma la expresión

$(a/b)\,\%m$ se sustituye por $(amodm * Bmodm)modm$ donde B es el inverso multiplicativo entre de b y m

3.4.5. Encontrar el inverso modular para una matriz de números módulo m

Supongamos que nos dan una matriz y queremos encontrar el inverso modular para todos los números que contiene (todos ellos son invertibles). En lugar de calcular el inverso de cada número, podemos expandir la fracción por el producto de prefijo (excluyéndose a sí mismo) y el producto de sufijo (excluyéndose a sí mismo), y terminar calculando solo un inverso.

$$\begin{aligned} x_i^{-1} &= \frac{1}{x_i} = \frac{\overbrace{x_1 \cdot x_2 \cdots x_{i-1}}^{\text{prefix}_{i-1}} \cdot 1 \cdot \overbrace{x_{i+1} \cdot x_{i+2} \cdots x_n}^{\text{suffix}_{i+1}}}{x_1 \cdot x_2 \cdots x_{i-1} \cdot x_i \cdot x_{i+1} \cdot x_{i+2} \cdots x_n} \\ &= \text{prefix}_{i-1} \cdot \text{suffix}_{i+1} \cdot (x_1 \cdot x_2 \cdots x_n)^{-1} \end{aligned}$$

En el código podemos simplemente hacer una matriz de productos de prefijo (excluirse, comenzar desde el elemento de identidad), calcular el inverso modular para el producto de todos los números y luego multiplicarlo por el producto de prefijo y sufijo (excluirse). El producto del sufijo se calcula iterando de atrás hacia adelante.

4. Implementación

Implementación del cálculo del inverso modular entre i y m usando la división euclidiana

4.1. C++

4.1.1. Encontrar el inverso modular utilizando el algoritmo euclidiano extendido

```
int extended_euclidean(int a, int b, int& x, int& y) {
   x = 1, y = 0;
   int x1 = 0, y1 = 1, a1 = a, b1 = b;
   while (b1) {
      int q = a1 / b1;
      tie(x, x1) = make_tuple(x1, x - q * x1);
      tie(y, y1) = make_tuple(y1, y - q * y1);
      tie(a1, b1) = make_tuple(b1, a1 - q * b1);
   }
   return a1;
}

int find_inverse_modular(int a, int m){
   int x, y;
   int g = extended_euclidean(a, m, x, y);
   if (g != 1) return -1;
   else return (x % m + m) % m;
}
```

4.1.2. Encontrar el inverso modular para módulos primos usando la división euclidiana

```
long long inverseModular(long long i, long long m) {
   return i <= 1 ? i : m - (long long)(m/i) * inv(m % i, m) % m;
}
```

4.1.3. Encontrar el inverso modular para todos los números de 1 a m

```
vector<int> inverse_modular_1_to_m(int m){
   vector<int> inv(m,0);
   inv[1] = 1;
   for(int a = 2; a < m; ++a) inv[a] = m - (m/a) * inv[m%a] % m;
   return inv;
}
```

4.1.4. Encontrar el inverso modular para una matriz de números módulo m

```
int extended_euclidean(int a, int b, int& x, int& y) {
   x = 1, y = 0;
   int x1 = 0, y1 = 1, a1 = a, b1 = b;
   while (b1) {
      int q = a1 / b1;
      tie(x, x1) = make_tuple(x1, x - q * x1);
      tie(y, y1) = make_tuple(y1, y - q * y1);
      tie(a1, b1) = make_tuple(b1, a1 - q * b1);
   }
   return a1;
}

vector<int> invs(const vector<int> &a, int m) {
   int n = a.size();
   if (n == 0) return {};
   vector<int> b(n);
   int v = 1;
   for (int i = 0; i != n; ++i) {
      b[i] = v;
      v = static_cast<long long>(v) * a[i] % m;
   }
   int x, y;
   extended_euclidean(v, m, x, y);
   x = (x % m + m) % m;
   for (int i = n - 1; i >= 0; --i) {
      b[i] = static_cast<long long>(x) * b[i] % m;
      x = static_cast<long long>(x) * a[i] % m;
   }
   return b;
}
```

4.2. Java

4.2.1. Encontrar el inverso modular utilizando el algoritmo euclidiano extendido

```
public static long[] extended_euclidean(long a, long b) {
   long x = 1, y = 0, x1 = 0, y1 = 1, t;
   while (b != 0) {
      long q = a / b; t = x;
      x = x1; x1 = t - q * x1;
      t = y; y = y1;
      y1 = t - q * y1; t = b;
      b = a - q * b; a = t;
   }
   return a > 0 ? new long[]{a, x, y} : new long[]{-a, -x, -y};
}


public static long find_inverse_modular(long a, long m){
   long [] sols = extended_euclidean(a, m);
   if (sols[0] != 1) return -1;
   else return (sols[1] % m + m) % m;
}
```

4.2.2. Encontrar el inverso modular para módulos primos usando la división euclidiana

```
public long inverseModular(long i, long m) {
   return i <= 1L ? i : m - (long)(m/i) * inv(m % i, m) % m;
}
```

4.2.3. Encontrar el inverso modular para todos los números de 1 a m

```
public static int [] inverse_modular_1_to_m(int m){
   int [] inv = new int [m];
   Arrays.fill(inv,0);
   inv[1] = 1;
   for(int a = 2; a < m; ++a) inv[a] = m - (m/a) * inv[m%a] % m;
   return inv;
}
```

4.2.4. Encontrar el inverso modular para una matriz de números módulo m

```
public static long[] extended_euclidean(long a, long b) {
```

```
    long x = 1, y = 0, x1 = 0, y1 = 1, t;
    while (b != 0) {
        long q = a / b; t = x;
        x = x1; x1 = t - q * x1;
        t = y; y = y1;
        y1 = t - q * y1; t = b;
        b = a - q * b; a = t;
    }
    return a > 0 ? new long[]{a, x, y} : new long[]{-a, -x, -y};
}

public static long [] invs(long [] a, long m) {
    int n = a.length;
    if (n == 0) return new long [0];
    long [] b =new long [n];
    long v = 1;
    for (int i = 0; i != n; ++i) {
        b[i] = v;
        v = v * a[i] % m;
    }
    long x, y;
    long [] sols =extended_euclidean(v, m);
    x= sols[1]; y= sols[2];
    x = (x % m + m) % m;
    for (int i = n - 1; i >= 0; --i) {
        b[i] = x * b[i] % m;
        x = x * a[i] % m;
    }
    return b;
}
```

5. Aplicaciones

La aritmética modular se utiliza sobre todo en ejercicios donde los valores crecen muy rápidos y pueden provocar desbordamiento producto a que los valores no pueden ser almacenados en por los tipos de datos definidos en los lenguajes de programación.

Cuando un problema te pide modular el resultado final es recomendable modular desde las operaciones intermedias aplicando lo visto hasta ahora para cada operación aritmética y de esta forma se evita posibles desbordamiento de datos.

Las aplicaciones de la aritmética modular son diversas y se encuentran en diferentes áreas, entre las cuales se destacan:

1. **Criptografía:** La aritmética modular es fundamental en la criptografía, ya que se utiliza para implementar algoritmos de cifrado y descifrado como el cifrado RSA, el algoritmo de intercambio de claves Diffie-Hellman, entre otros. La seguridad de estos algoritmos se basa

en la dificultad computacional de resolver ciertos problemas relacionados con la aritmética modular.

2. **Teoría de números:** La aritmética modular es una herramienta importante en la teoría de números para el estudio de propiedades de los números enteros, como la congruencia, los residuos cuadráticos, el teorema chino del resto, entre otros.
3. **Computación y programación:** En informática, la aritmética modular se utiliza en la implementación de algoritmos eficientes para realizar operaciones aritméticas en números grandes. Por ejemplo, en la programación competitiva y en la optimización de cálculos numéricos.
4. **Matemáticas aplicadas:** La aritmética modular tiene aplicaciones en diversas áreas de las matemáticas aplicadas, como la teoría de grafos, la teoría de códigos correctores de errores, la teoría de autómatas finitos, entre otras.

6. Complejidad

Encontrar el inverso modular utilizando el algoritmo euclidiano extendido tiene una complejidad propia de la implementación del algoritmo euclidiano extendido, es decir O($\log \min(a, b)$).

Aunque encontrar el inverso modular usando exponenciación binaria es más fácil de entender que el utilizando el algoritmo euclidiano extendido, en el caso de que m no es un número primo, necesitamos calcular la función phi de Euler, que implica la factorización de m, lo que podría resultar muy difícil. Si la factorización prima de m se conoce, entonces la complejidad de este método es $O(\log m)$.

La complejidad temporal exacta de la recursión utilizada para Encontrar el inverso modular para módulos primos usando la división euclidiana no se conoce. Está en algún lugar entre $O(\frac{\log m}{\log \log m})$ y $O(n^{\frac{1}{3}-\frac{2}{177}+\epsilon})$. En la práctica, esta implementación es rápida, por ejemplo, para el módulo $10^9 + 7$ siempre terminará en menos de 50 iteraciones.

Para calcular previamente el inverso modular para cada número en el rango $[1, m - 1]$ se hace con una complejidad de $O(m)$.

7. Ejercicios

La siguiente lista de ejercicios utilizan la aritmética modular para dar la solución final.

- DMOJ - Last Digit of A^B
- DMOJ - Fibonacci Calculation
- DMOJ - Fibonacci 2D
- DMOJ - Último dígito
- DMOJ -Tiling

- DMOJ - Zapis
- UVa 11904 - One Unit Machine
- Hackerrank - Longest Increasing Subsequence Arrays
- Codeforces 300C - Beautiful Numbers
- Codeforces 622F - The Sum of the k-th Powers
- Codeforces 717A - Festival Organization
- Codeforces 896D - Nephren Runs a Cinema