Primalidad y factorización

Agustín Santiago Gutiérrez

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Campamento Caribeño ACM-ICPC 2016

Contenidos

Contenidos

Definición

Aritmética módulo M ($\mathbb{Z}_M$)

La aritmética módulo M consiste en una modificación de la aritmética usual de números enteros, en la cual trabajamos únicamente con el *resto* de los números al ser divididos por un cierto entero fijo $M > 0$, ignorando "todo lo demás" de los números involucrados.

Definición

Aritmética módulo M ($\mathbb{Z}_M$)

La aritmética módulo M consiste en una modificación de la aritmética usual de números enteros, en la cual trabajamos únicamente con el *resto* de los números al ser divididos por un cierto entero fijo $M > 0$, ignorando "todo lo demás" de los números involucrados.

- Así, $11 = 18$ si estamos trabajando módulo 7, pues ambos dejan un resto de 4 en la división por 7. Esto se suele notar $11 \equiv 18 (\text{mód } 7)$

Definición

Aritmética módulo M ($\mathbb{Z}_M$)

La aritmética módulo M consiste en una modificación de la aritmética usual de números enteros, en la cual trabajamos únicamente con el *resto* de los números al ser divididos por un cierto entero fijo $M > 0$, ignorando "todo lo demás" de los números involucrados.

- Así, $11 = 18$ si estamos trabajando módulo 7, pues ambos dejan un resto de 4 en la división por 7. Esto se suele notar $11 \equiv 18 (\text{mód } 7)$
- Una forma operacional de ver esta aritmética es suponer que todo el tiempo tenemos los números reducidos al rango de enteros en $[0, M)$, y tomamos el resto de la división por M para devolverlos a ese rango luego de cada operación.

Propiedades

A los efectos de realizar sumas, restas y productos, la aritmética módular es análoga a la aritmética usual, manteniendo sus propiedades importantes.

- $a + b \equiv b + a (\text{mód } M)$
- $(a + b) + c \equiv a + (b + c) (\text{mód } M)$
- 0 es el neutro de la suma.
- Para todo a existe un único inverso aditivo modular $-a$, $a + (-a) \equiv 0 (\text{mód } M)$. $a - b \equiv a + (-b) (\text{mód } M)$
- $a \cdot b \equiv b \cdot a (\text{mód } M)$
- $(a \cdot b) \cdot c \equiv a \cdot (b \cdot c) (\text{mód } M)$
- 1 es el neutro del producto.
- $(a + b) \cdot c \equiv a \cdot c + b \cdot c (\text{mód } M)$

Forma operacional en el código

Supongamos que se tiene que computar una suma de los números enteros `a[0]` hasta `a[N-1]`, pero solamente nos importan los últimos 4 dígitos (equivale a trabajar módulo 10000).

```
1 int result = 0;
2 for (int i = 0; i < N; i++)
3     result = (result + a[i]) %10000;
```

Como decíamos antes, a nivel de operaciones trabajar con aritmética modular equivale a simplemente tomar módulo luego de cada operación aritmética básica.

Problema ante números negativos

Sin embargo, la implementación anterior puede resultar problemática al trabajar con números **negativos**.

- Si por ejemplo fuera $N = 2$, `a[0]=123` y `a[1]=-200`, el código anterior produce -77, que puede no ser lo deseado.
- Incluso si no hay números negativos en el problema, es muy común que **restemos** números en nuestra solución.
- Estos resultados con valores negativos ocurren porque el resultado de la división entera se redondea hacia cero en los lenguajes y plataformas más populares.

Problema ante números negativos

Sin embargo, la implementación anterior puede resultar problemática al trabajar con números **negativos**.

- Si por ejemplo fuera $N = 2$, `a[0]=123` y `a[1]=-200`, el código anterior produce -77, que puede no ser lo deseado.
- Incluso si no hay números negativos en el problema, es muy común que **restemos** números en nuestra solución.
- Estos resultados con valores negativos ocurren porque el resultado de la división entera se redondea hacia cero en los lenguajes y plataformas más populares.
- Solución:

```
int MOD(int x, int M){return ((x%M)+M)%M;}
```

Cuidado con el overflow

- Otro problema al que es especialmente común enfrentarse al trabajar con aritmética modular es el peligro de tener overflow en las operaciones.
- Por esto es que tomamos módulo luego de cada operación, y no solamente al final de todo el programa.
- Truquito en C++: Tener en cuenta el tipo `__int128`, entero de 128 bits. No está presente en todo judge, pero puede ser muy útil cuando está disponible.

¿Qué pasa con la división?

- ¿Podemos operar modularmente con la división tal cual lo hacemos con sumas, restas y productos?

¿Qué pasa con la división?

- ¿Podemos operar modularmente con la división tal cual lo hacemos con sumas, restas y productos?
- **NO**. Por ejemplo:
  $\frac{10}{2} \equiv 5(\text{mód } 8)$, pero $10 \equiv 2(\text{mód } 8)$ y $\frac{2}{2} \equiv 1 \not\equiv 5(\text{mód } 8)$
- Podemos garantizar que este "truco" funciona cuando el módulo es un número **primo**.
  $\frac{27}{3} \equiv 2(\text{mód } 7)$, $27 \equiv 6(\text{mód } 7)$ y $\frac{6}{3} \equiv 2(\text{mód } 7)$
  Pero solo si el divisor **no es cero** (módulo p)
  $\frac{140}{14} \equiv 3(\text{mód } 7)$, pero $140 \equiv 14 \equiv 0(\text{mód } 7)$ y $\frac{0}{0} \equiv ?(\text{mód } 7)$

¿Qué pasa con la división?

- ¿Podemos operar modularmente con la división tal cual lo hacemos con sumas, restas y productos?
- **NO**. Por ejemplo:
  $\frac{10}{2} \equiv 5(\text{mód } 8)$, pero $10 \equiv 2(\text{mód } 8)$ y $\frac{2}{2} \equiv 1 \not\equiv 5(\text{mód } 8)$
- Podemos garantizar que este "truco" funciona cuando el módulo es un número **primo**.
  $\frac{27}{3} \equiv 2(\text{mód } 7)$, $27 \equiv 6(\text{mód } 7)$ y $\frac{6}{3} \equiv 2(\text{mód } 7)$
  Pero solo si el divisor **no es cero** (módulo p)
  $\frac{140}{14} \equiv 3(\text{mód } 7)$, pero $140 \equiv 14 \equiv 0(\text{mód } 7)$ y $\frac{0}{0} \equiv ?(\text{mód } 7)$
- ¿Pero y si aún con un módulo primo, la división modular no resulta una división entera?
  $\frac{12}{3} \equiv 4(\text{mód } 7)$, pero $12 \equiv 5(\text{mód } 7)$ y $\frac{5}{3} \equiv ?(\text{mód } 7)$

Inversos modulares

Definición

Decimos que b es inverso de a módulo M si $a \cdot b \equiv 1 (\text{mód } M)$.

Notar que solo un $a \not\equiv 0 (\text{mód } M)$ podría tener un inverso, y de existir el inverso es único, y a su vez a resulta ser el inverso de b.

Inversos modulares

Definición

Decimos que b es inverso de a módulo M si $a \cdot b \equiv 1 (\text{mód } M)$.

Notar que solo un $a \not\equiv 0 (\text{mód } M)$ podría tener un inverso, y de existir el inverso es único, y a su vez a resulta ser el inverso de b.

Teorema

Si p es un número primo, entonces todo número $a \not\equiv 0 (\text{mód } p)$ tiene un inverso módulo p.

Inversos modulares: utilidad

- Recordemos que para realizar $3/2 = 1{,}5$, en realidad podríamos multiplicar directamente por el inverso de 2, es decir $3 \cdot 0{,}5 = 1{,}5$
- Lo mismo podemos hacer modularmente. Por ejemplo, $5 \cdot 3 \equiv 1 (\text{mód } 7)$, así que $inv(3) = 5$. Y entonces recordando el ejemplo anterior:
  $\frac{12}{3} \equiv 4 (\text{mód } 7)$,
  $12 \equiv 5 (\text{mód } 7)$ y
  "$\frac{5}{3}$" $\equiv 5 \cdot inv(3) \equiv 5 \cdot 5 \equiv 4 (\text{mód } 7)$
- De esta forma, ya podemos dividir modularmente al trabajar con un número primo (excepto cuando el divisor se hace 0 módulo p).
- ¿Pero cómo calculamos los inversos?

Contenidos

Pequeño teorema de Fermat

Teorema

Si p es primo y $a \not\equiv 0 (\text{mód } p)$, entonces $a^{p-1} \equiv 1 (\text{mód } p)$

- Por ejemplo $6^{30} = 713141676518494702 9025 \cdot 31 + 1$
- ¿Para qué puede servir este teorema?

Aplicación 1: Cálculo de inversos

- Recordemos que dado $a \not\equiv 0$, si encontramos algún número x tal que $a \cdot x \equiv 1 (\text{mód } p)$, x será automáticamente el inverso de a.
- Si tomamos $x = a^{p-2}$, ¿Cuánto vale $a \cdot x$?

Aplicación 1: Cálculo de inversos

- Recordemos que dado $a \not\equiv 0$, si encontramos algún número x tal que $a \cdot x \equiv 1 (\text{mód } p)$, x será automáticamente el inverso de a.
- Si tomamos $x = a^{p-2}$, ¿Cuánto vale $a \cdot x$?
- Tenemos $a \cdot x = a^{p-1} \equiv 1 (\text{mód } p)$ por el Pequeño Teorema de Fermat.
- Luego para cada a su inverso será simplemente a^{p-2}.

Aplicación 2: Testeo de residuo cuadrático

Definición

Un resto r se dice un *residuo cuadrático* módulo p si existe x tal que $x^2 \equiv r (\text{mód } p)$

Por ejemplo los residuos cuadráticos módulo 5 son $0, 1, 4$. Notar que 0 siempre es residuo cuadrático módulo p.

- Si $r \not\equiv 0$ es residuo cuadrático, ¿Cuánto vale $r^{\frac{p-1}{2}}$?

Aplicación 2: Testeo de residuo cuadrático

Definición

Un resto r se dice un *residuo cuadrático* módulo p si existe x tal que $x^2 \equiv r (\text{mód } p)$

Por ejemplo los residuos cuadráticos módulo 5 son $0, 1, 4$. Notar que 0 siempre es residuo cuadrático módulo p.

- Si $r \not\equiv 0$ es residuo cuadrático, ¿Cuánto vale $r^{\frac{p-1}{2}}$?
- $r \equiv x^2$ para algún $x \not\equiv 0$, y entonces $r^{\frac{p-1}{2}} \equiv (x^2)^{\frac{p-1}{2}} \equiv 1 (\text{mód } p)$
- Se puede verificar que además si para algún r vale $r^{\frac{p-1}{2}} \equiv 1 (\text{mód } p)$, r es residuo cuadrático módulo p.

Contenidos

Potenciación logarítmica

- En los ejemplos anteriores hemos reducido algunos problemas a calcular a^b módulo M, para enteros no negativos a, b, M.
- ¿Cómo hacemos esto más eficientemente que realizando $b - 1$ multiplicaciones?

Potenciación logarítmica (idea)

- Una buena idea es pensar en ir elevando al cuadrado sucesivamente, lo cual permite que el exponente crezca rápidamente.
- Pensado de manera recursiva, si llamamos $f(a, n) = a^n$:

$$f(a,n) = \begin{cases} 1 & \text{si } n = 0 \\ f\left(a^2, \frac{n}{2}\right) & \text{si } n \text{ es par} \\ a \cdot f\left(a^2, \left\lfloor \frac{n}{2} \right\rfloor\right) & \text{si } n \text{ es impar} \end{cases}$$

Potenciación logarítmica (código)

```
1  typedef long long tint;
2  tint potlog(tint a,tint b, const tint M)
3  {
4      tint res = 1;
5      while (b > 0)
6      {
7          if (b%2 != 0)
8              res = MOD(res*a, M);
9          a = MOD(a*a, M);
10         b /= 2;
11     }
12     return res;
13 }
```

Invariante de ciclo:
La respuesta que deseamos es $res \cdot (a^b \text{ módulo } M)$
Este método realiza solamente $O(\lg b)$ multiplicaciones.

Contenidos

Problema

- Una necesidad muy usual al trabajar con números primos es la de calcular *todos* los primos desde 1 hasta N para un cierto N.
- La idea es realizarlo de manera más eficiente que verificando la primalidad de cada número por separado.

Idea 1

- Si verificásemos cada número por separado, lo que haríamos sería recorrer sus posibles **divisores** para ver si es primo.
- ¿Que pasaría si en lugar de probar los divisores de cada número, descartásemos sus **múltiplos**?

Idea 1

- Si verificásemos cada número por separado, lo que haríamos sería recorrer sus posibles **divisores** para ver si es primo.
- ¿Que pasaría si en lugar de probar los divisores de cada número, descartásemos sus **múltiplos**?
- Hay $\frac{N}{1}$ múltiplos de 1, $\frac{N}{2}$ múltiplos de 2, $\cdots$ $\frac{N}{N}$ múltiplos de N. Luego si recorremos todo el costo total es

$$\sum_{i=1}^{N} \frac{N}{i} = N \sum_{i=1}^{N} \frac{1}{i} = \Theta(N \lg N)$$

Idea 1 (cont)

- Esta idea es verdaderamente muy sencilla, y como vimos ya alcanza una eficiencia aceptable.
- Además, al recorrer los múltiplos de todos los números, se encuentran todos los divisores propios de todos los números.
- De esta forma es extremadamente fácil modificar esta versión para computar fácilmente sumas de divisores, cantidades, y otras funciones similares.

```
1  for(int i = 0; i < MAX; i++) p[i] = true;
2  p[0] = p[1] = false;
3  for (int i = 2; i < MAX; i++)
4      for (int j = 2*i; j < MAX; j += i) p[j] = false;
```

Idea 2

- Todo número compuesto tiene un divisor **primo**.
- Por lo tanto, alcanza con descartar los múltiplos de los números primos que vamos encontrando.
- La cantidad de operaciones a realizar en este caso se reduce a $\Theta(N \lg \lg N)$, que es “casi lineal”.

```
1    for(int i = 0; i < MAX; i++) p[i] = true;
2    p[0] = p[1] = false;
3    for (int i = 2; i < MAX; i++)
4    if (p[i])
5        for (int j = 2*i; j < MAX; j += i) p[j] = false;
```

Idea 3

- Todo número compuesto tiene un divisor primo p, con $p \leq \sqrt{N}$.
- Podemos parar el proceso de descarte de múltiplos en $\sqrt{N}$.
- La cantidad de operaciones sigue siendo $\Theta(N \lg \lg N)$.

```
1  for(int i = 0; i < MAX; i++) p[i] = true;
2  p[0] = p[1] = false;
3  for (int i = 2; i*i < MAX; i++)
4  if (p[i])
5      for (int j = 2*i; j < MAX; j += i) p[j] = false;
```

Idea 4

- De manera similar al caso anterior, si $N < p^2$ es compuesto, tiene un divisor primo menor que p, y ya habrá sido descartado.
- Podemos comenzar el descarte de los múltiplos de i por i^2.
- Aún con esta optimización, la cantidad de operaciones sigue siendo $\Theta(N \lg \lg N)$.

```
1    for(int i = 0; i < MAX; i++) p[i] = true;
2    p[0] = p[1] = false;
3    for (int i = 2; i*i < MAX; i++)
4    if (p[i])
5        for (int j = i*i; j < MAX; j += i) p[j] = false;
```

Tiempos

Para `MAX=300M`:

`RecorrerMultiplos`	39.1 s
`CribaBasica`	5.6 s
`CribaHastaRaiz`	3.2 s
`CribaHastaRaizDesdeICuadrado`	3.0 s

Contenidos

Contenidos

Contenidos

Contenidos

Contenidos

Contenidos

Contenidos