

# Teoría de números

---

Miguel Ortiz

**Programación competitiva para ICPC**

Mayo 2023 - Cochabamba, Bolivia

¿Cuál es la menor potencia de 2 que es divisible entre 3?

# Introducción

- Es un área de las matemáticas que estudia las propiedades de los números enteros
- Los números primos son muy importantes para su análisis

# Introducción

- Es un área de las matemáticas que estudia las propiedades de los números enteros
- Los números primos son muy importantes para su análisis
- Se dice que un número es primo si es un número natural mayor a 1 y no es representable como el producto de dos naturales menores a él
- Solo tiene dos divisores: 1 y él mismo

# Introducción

- Es un área de las matemáticas que estudia las propiedades de los números enteros
- Los números primos son muy importantes para su análisis
- Se dice que un número es primo si es un número natural mayor a 1 y no es representable como el producto de dos naturales menores a él
- Solo tiene dos divisores: 1 y él mismo
- Los números no primos se llaman *compuestos*

# Teorema fundamental de la aritmética

Todo entero mayor a 1 puede ser representado como un **único** producto de números primos

$$n = p_1^{n_1} p_2^{n_2} \cdots p_k^{n_k} = \prod_{i=1}^k p_i^{n_i}$$

# Teorema fundamental de la aritmética

Todo entero mayor a 1 puede ser representado como un **único** producto de números primos

$$n = p_1^{n_1} p_2^{n_2} \cdots p_k^{n_k} = \prod_{i=1}^k p_i^{n_i}$$

Por ejemplo:

- $999 = 2^3 \cdot 37$
- $1000 = 2^3 \cdot 5^3$
- $1001 = 7 \cdot 11 \cdot 13$

# Encontrar divisores

- Se puede iterar por todos los números entre 1 y  $n$  y verificar si el residuo de la división es 0

```
for (int i = 1; i <= n; i++) {  
    if (n % i == 0) {  
        // i es divisor de n  
    }  
}
```

- $O(n)$



# Encontrar divisores

- Si  $d$  divide a  $n$ , entonces  $\frac{n}{d}$  también divide a  $n$
- Si revisamos todos los valores  $d$  tal que  $d \leq \frac{n}{d}$ , solo necesitamos iterar hasta  $\sqrt{n}$

```
for (int i = 1; i*i <= n; i++) {  
    if (n % i == 0) {  
        // i es divisor de n  
        // n/i es divisor de n  
    }  
}
```

- $O(\sqrt{n})$

# Encontrar divisores

- Si  $d$  divide a  $n$ , entonces  $\frac{n}{d}$  también divide a  $n$
- Si revisamos todos los valores  $d$  tal que  $d \leq \frac{n}{d}$ , solo necesitamos iterar hasta  $\sqrt{n}$

```
for (int i = 1; i*i <= n; i++) {  
    if (n % i == 0) {  
        // i es divisor de n  
        // n/i es divisor de n  
    }  
}
```

- $O(\sqrt{n})$
- Podemos usar esto para verificar si un número es primo

# Criba de Eratóstenes

- Es posible encontrar todos los primos de 1 a  $n$  en  $O(n\sqrt{n})$  con el anterior método
- La criba de Eratóstenes encuentra todos los primos de 1 a  $n$  en  $O(n \log \log n)$

# Criba de Eratóstenes

- Es posible encontrar todos los primos de 1 a  $n$  en  $O(n\sqrt{n})$  con el anterior método
- La criba de Eratóstenes encuentra todos los primos de 1 a  $n$  en  $O(n \log \log n)$
- El algoritmo construye un arreglo `criba` marcando todos los números compuestos
  - `criba[i]` es `true` si  $i$  es compuesto
  - `criba[i]` es `false` si  $i$  es primo

# Criba de Eratóstenes

- Es posible encontrar todos los primos de 1 a  $n$  en  $O(n\sqrt{n})$  con el anterior método
- La criba de Eratóstenes encuentra todos los primos de 1 a  $n$  en  $O(n \log \log n)$
- El algoritmo construye un arreglo `criba` marcando todos los números compuestos
  - `criba[i]` es `true` si  $i$  es compuesto
  - `criba[i]` es `false` si  $i$  es primo
- Se revisan los números de 2 a  $n$ , si el número es primo, se marcan todos sus múltiplos como compuestos

# Criba de Eratóstenes

```
vector<bool> criba(n+1, false);
criba[0] = criba[1] = true; // 0 y 1 no son primos
for (int i = 2; i <= n; i++) {
    if (!criba[i]) {
        // i es primo
        for (int j = 2*i; j <= n; j += i) {
            criba[j] = true;
        }
    }
}
```

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30

# Criba de Eratóstenes

```
vector<bool> criba(n+1, false);
criba[0] = criba[1] = true; // 0 y 1 no son primos
for (int i = 2; i <= n; i++) {
    if (!criba[i]) {
        // i es primo
        for (int j = 2*i; j <= n; j += i) {
            criba[j] = true;
        }
    }
}
```

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30

# Criba de Eratóstenes

```
vector<bool> criba(n+1, false);
criba[0] = criba[1] = true; // 0 y 1 no son primos
for (int i = 2; i <= n; i++) {
    if (!criba[i]) {
        // i es primo
        for (int j = 2*i; j <= n; j += i) {
            criba[j] = true;
        }
    }
}
```

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30



# Criba de Eratóstenes

```
vector<bool> criba(n+1, false);
criba[0] = criba[1] = true; // 0 y 1 no son primos
for (int i = 2; i <= n; i++) {
    if (!criba[i]) {
        // i es primo
        for (int j = 2*i; j <= n; j += i) {
            criba[j] = true;
        }
    }
}
```

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30

# Criba de Eratóstenes

```
vector<bool> criba(n+1, false);  
criba[0] = criba[1] = true; // 0 y 1 no son primos  
for (int i = 2; i <= n; i++) {  
    if (!criba[i]) {  
        // i es primo  
        for (int j = 2*i; j <= n; j += i) {  
            criba[j] = true;  
        }  
    }  
}
```

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30

# Criba de Eratóstenes

```
vector<bool> criba(n+1, false);
criba[0] = criba[1] = true; // 0 y 1 no son primos
for (int i = 2; i <= n; i++) {
    if (!criba[i]) {
        // i es primo
        for (int j = 2*i; j <= n; j += i) {
            criba[j] = true;
        }
    }
}
```

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30

# Criba de Eratóstenes

```
vector<bool> criba(n+1, false);
criba[0] = criba[1] = true; // 0 y 1 no son primos
for (int i = 2; i <= n; i++) {
    if (!criba[i]) {
        // i es primo
        for (int j = 2*i; j <= n; j += i) {
            criba[j] = true;
        }
    }
}
```

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30

# Encontrar factores primos

- Queremos encontrar los factores primos de un número  $x$
- Guardamos los primos en un vector mientras realizamos la criba
- El exponente de  $p$  es igual a la cantidad de veces que  $p$  divide a  $x$

```
map<int, int> factorizacion;
for (int p : primos) if (x % p == 0) {
    int e = 0;
    while (x % p == 0) {
        x /= p;
        e++;
    }
    factorizacion[p] = e;
}
if (x > 1) factorizacion[x] = 1;
```

# Encontrar factores primos

- Queremos encontrar los factores primos de un número  $x$
- Guardamos los primos en un vector mientras realizamos la criba
- El exponente de  $p$  es igual a la cantidad de veces que  $p$  divide a  $x$

```
map<int, int> factorizacion;
for (int p : primos) if (x % p == 0) {
    int e = 0;
    while (x % p == 0) {
        x /= p;
        e++;
    }
    factorizacion[p] = e;
}
if (x > 1) factorizacion[x] = 1;
```

- Si generamos la criba hasta  $n$ , funciona para  $x \leq n^2$  en  $O(\pi(n))$

- Greatest Common Divisor (Máximo Común Divisor)
  - Dados dos números  $a$  y  $b$ , su GCD es el mayor número que divide a ambos
- Least Common Multiple (Mínimo Común Múltiplo)
  - Dados dos números  $a$  y  $b$ , su LCM es el menor número que es múltiplo de ambos

# GCD y LCM

- Podemos calcular el GCD y LCM de dos números usando sus factorizaciones
- $p^a \cdot p^b = p^{a+b}$
- $p^a / p^b = p^{a-b}$



- Podemos calcular el GCD y LCM de dos números usando sus factorizaciones
- $p^a \cdot p^b = p^{a+b}$
- $p^a / p^b = p^{a-b}$
- Si  $d$  es un divisor de  $n$ ,  $\forall p_i$  en la factorización de  $n$ , el exponente de  $p_i$  en  $d$  es menor o igual que el exponente de  $p_i$  en  $n$
- El exponente debe ser mayor o igual si  $d$  es un múltiplo de  $n$

- Podemos calcular el GCD y LCM de dos números usando sus factorizaciones
- $p^a \cdot p^b = p^{a+b}$
- $p^a / p^b = p^{a-b}$
- Si  $d$  es un divisor de  $n$ ,  $\forall p_i$  en la factorización de  $n$ , el exponente de  $p_i$  en  $d$  es menor o igual que el exponente de  $p_i$  en  $n$
- El exponente debe ser mayor o igual si  $d$  es un múltiplo de  $n$
- $\gcd(a, b)$  es el número cuya factorización tiene los exponentes mínimos entre  $a$  y  $b$  por cada primo
- Similarmente,  $\text{lcm}(a, b)$  tiene los exponentes máximos

# GCD y LCM

- Podemos calcular el GCD y LCM de dos números usando sus factorizaciones
- $p^a \cdot p^b = p^{a+b}$
- $p^a / p^b = p^{a-b}$
- Si  $d$  es un divisor de  $n$ ,  $\forall p_i$  en la factorización de  $n$ , el exponente de  $p_i$  en  $d$  es menor o igual que el exponente de  $p_i$  en  $n$
- El exponente debe ser mayor o igual si  $d$  es un múltiplo de  $n$
- $\gcd(a, b)$  es el número cuya factorización tiene los exponentes mínimos entre  $a$  y  $b$  por cada primo
- Similarmente,  $\text{lcm}(a, b)$  tiene los exponentes máximos
- $\gcd(2^3 \cdot 3^2 \cdot 5^1, 2^2 \cdot 3^4 \cdot 7^1) = 2^2 \cdot 3^2$
- $\text{lcm}(2^3 \cdot 3^2 \cdot 5^1, 2^2 \cdot 3^4 \cdot 7^1) = 2^3 \cdot 3^4 \cdot 5^1 \cdot 7^1$

- Este método es bueno para pensar en soluciones o demostrarlas, pero no es eficiente y es tedioso de implementar
- Se pueden usar las funciones ya presentes en los lenguajes para calcular el GCD y LCM

- Este método es bueno para pensar en soluciones o demostrarlas, pero no es eficiente y es tedioso de implementar
- Se pueden usar las funciones ya presentes en los lenguajes para calcular el GCD y LCM

```
--gcd(a, b) // C++
```

```
import math
```

```
math.gcd(a, b) # Python
```

- Este método es bueno para pensar en soluciones o demostrarlas, pero no es eficiente y es tedioso de implementar
- Se pueden usar las funciones ya presentes en los lenguajes para calcular el GCD y LCM

```
--gcd(a, b) // C++
```

```
import math
```

```
math.gcd(a, b) # Python
```

- Para calcular el LCM, podemos usar la fórmula  $\text{lcm}(a, b) = \frac{a \cdot b}{\text{gcd}(a, b)}$

# Aritmética modular

---

- Recordemos que  $\frac{a}{m} \rightarrow a = m \cdot c + r$ , siendo  $c$  el cociente y  $r$  el residuo
- La aritmética modular trabaja solo con los residuos
- $a \bmod m \rightarrow$  residuo de dividir  $a$  entre  $m$



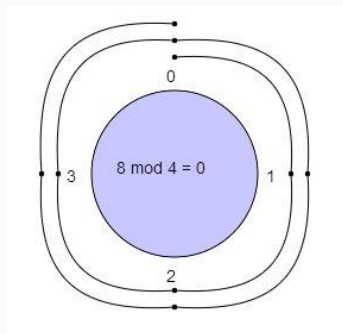
- Es útil visualizar los residuos como posiciones en un reloj de tamaño  $m$
- Observemos lo que pasa aumentando valores de 1 en 1 con división entre 3
  - $0/3 = 0$  residuo 0
  - $1/3 = 0$  residuo 1
  - $2/3 = 0$  residuo 2
  - $3/3 = 1$  residuo 0
  - $4/3 = 1$  residuo 1
  - $5/3 = 1$  residuo 2
  - $6/3 = 2$  residuo 0

- Es útil visualizar los residuos como posiciones en un reloj de tamaño  $m$
- Observemos lo que pasa aumentando valores de 1 en 1 con división entre 3
  - $0/3 = 0$  residuo 0
  - $1/3 = 0$  residuo 1
  - $2/3 = 0$  residuo 2
  - $3/3 = 1$  residuo 0
  - $4/3 = 1$  residuo 1
  - $5/3 = 1$  residuo 2
  - $6/3 = 2$  residuo 0
- Los residuos aumentan de 1 en 1 hasta llegar al módulo menos 1, luego vuelven a 0 y se repiten

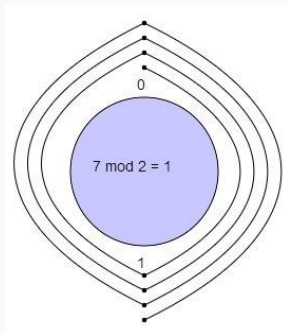
- Es útil visualizar los residuos como posiciones en un reloj de tamaño  $m$
- Observemos lo que pasa aumentando valores de 1 en 1 con división entre 3
  - $0/3 = 0$  residuo 0
  - $1/3 = 0$  residuo 1
  - $2/3 = 0$  residuo 2
  - $3/3 = 1$  residuo 0
  - $4/3 = 1$  residuo 1
  - $5/3 = 1$  residuo 2
  - $6/3 = 2$  residuo 0
- Los residuos aumentan de 1 en 1 hasta llegar al módulo menos 1, luego vuelven a 0 y se repiten
- Empezar en la posición 0 del reloj y avanzar  $x$  posiciones es equivalente a calcular  $x \bmod m$

# Aritmética modular

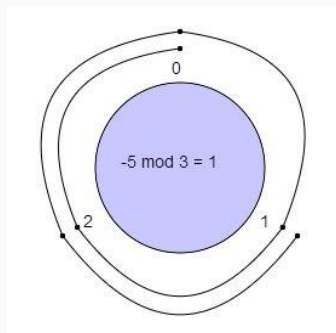
$$8 \bmod 4 = 0$$



$$7 \bmod 2 = 1$$



$$-5 \bmod 3 = 1$$



- **Congruencia:**  $a \equiv b \pmod{m} \rightarrow a \pmod{m} = b \pmod{m}$

- Veremos las siguientes operaciones como se harían en un lenguaje de programación

```
int r = a % m; // C++
```

```
r = a % m # Python
```



- Veremos las siguientes operaciones como se harían en un lenguaje de programación

```
int r = a % m; // C++
```

```
r = a % m # Python
```

- **Suma:**  $(a + b) \% m = (a \% m + b \% m) \% m$ , avanzar posiciones en el reloj

- Veremos las siguientes operaciones como se harían en un lenguaje de programación

```
int r = a % m; // C++
```

```
r = a % m # Python
```

- **Suma:**  $(a + b) \% m = (a \% m + b \% m) \% m$ , avanzar posiciones en el reloj
- **Resta:**  $(a - b) \% m = (a \% m - b \% m + m) \% m$ , cuidado con negativos

- **Multipliación:**  $(a \cdot b) \% m = (a \% m \cdot b \% m) \% m$

- **Multipliación:**  $(a \cdot b) \% m = (a \% m \cdot b \% m) \% m$
- ¿Por qué?

- **Multipliación:**  $(a \cdot b) \% m = (a \% m \cdot b \% m) \% m$
- ¿Por qué?
- Desarrollando:

- **Multipliación:**  $(a \cdot b) \% m = (a \% m \cdot b \% m) \% m$
- ¿Por qué?
- Desarrollando:
  1.  $((m \cdot c_a + r_a) \cdot (m \cdot c_b + r_b)) \% m$

- **Multipliación:**  $(a \cdot b) \% m = (a \% m \cdot b \% m) \% m$
- ¿Por qué?
- Desarrollando:
  1.  $((m \cdot c_a + r_a) \cdot (m \cdot c_b + r_b)) \% m$
  2.  $(m^2 \cdot c_a \cdot c_b + m \cdot c_a \cdot r_b + m \cdot c_b \cdot r_a + r_a \cdot r_b) \% m$

- **Multipliación:**  $(a \cdot b) \% m = (a \% m \cdot b \% m) \% m$
- ¿Por qué?
- Desarrollando:
  1.  $((m \cdot c_a + r_a) \cdot (m \cdot c_b + r_b)) \% m$
  2.  $(m^2 \cdot c_a \cdot c_b + m \cdot c_a \cdot r_b + m \cdot c_b \cdot r_a + r_a \cdot r_b) \% m$   
Repartimos el módulo en la suma
  3.  $((m^2 \cdot c_a \cdot c_b \% m) + (m \cdot c_a \cdot r_b \% m) + (m \cdot c_b \cdot r_a \% m) + (r_a \cdot r_b \% m)) \% m$



- **Multipliación:**  $(a \cdot b) \% m = (a \% m \cdot b \% m) \% m$
- ¿Por qué?
- Desarrollando:
  1.  $((m \cdot c_a + r_a) \cdot (m \cdot c_b + r_b)) \% m$
  2.  $(m^2 \cdot c_a \cdot c_b + m \cdot c_a \cdot r_b + m \cdot c_b \cdot r_a + r_a \cdot r_b) \% m$   
Repartimos el módulo en la suma
  3.  $((m^2 \cdot c_a \cdot c_b \% m) + (m \cdot c_a \cdot r_b \% m) + (m \cdot c_b \cdot r_a \% m) + (r_a \cdot r_b \% m)) \% m$
  4.  $(0 + 0 + 0 + (r_a \cdot r_b \% m)) \% m$

- **Multipliación:**  $(a \cdot b) \% m = (a \% m \cdot b \% m) \% m$
- ¿Por qué?
- Desarrollando:
  1.  $((m \cdot c_a + r_a) \cdot (m \cdot c_b + r_b)) \% m$
  2.  $(m^2 \cdot c_a \cdot c_b + m \cdot c_a \cdot r_b + m \cdot c_b \cdot r_a + r_a \cdot r_b) \% m$   
Repartimos el módulo en la suma
  3.  $((m^2 \cdot c_a \cdot c_b \% m) + (m \cdot c_a \cdot r_b \% m) + (m \cdot c_b \cdot r_a \% m) + (r_a \cdot r_b \% m)) \% m$
  4.  $(0 + 0 + 0 + (r_a \cdot r_b \% m)) \% m$
  5.  $r_x = x \% m \rightarrow (a \% m \cdot b \% m) \% m$

- **División:**  $(a/b)\%m = (a\%m \cdot b^{-1}\%m)\%m$
- No podemos repartir en la división
- Ej:  $\frac{21}{7} \bmod 5 \neq \frac{1}{2}$

- **División:**  $(a/b)\%m = (a\%m \cdot b^{-1}\%m)\%m$
- No podemos repartir en la división
- Ej:  $\frac{21}{7} \bmod 5 \neq \frac{1}{2}$
  
- $a^{-1} \bmod m$  es el inverso modular de  $a$  módulo  $m$
- ¿Cómo obtenemos  $a^{-1} \bmod m$ ?

- **División:**  $(a/b)\%m = (a\%m \cdot b^{-1}\%m)\%m$
- No podemos repartir en la división
- Ej:  $\frac{21}{7} \bmod 5 \neq \frac{1}{2}$
  
- $a^{-1} \bmod m$  es el inverso modular de  $a$  módulo  $m$
- ¿Cómo obtenemos  $a^{-1} \bmod m$ ?
- Pequeño teorema de Fermat:  $a^{m-1} \equiv 1 \bmod m$ , si  $m$  es primo y  $a$  no es múltiplo de  $m$

- **División:**  $(a/b)\%m = (a\%m \cdot b^{-1}\%m)\%m$
- No podemos repartir en la división
- Ej:  $\frac{21}{7} \bmod 5 \neq \frac{1}{2}$
  
- $a^{-1} \bmod m$  es el inverso modular de  $a$  módulo  $m$
- ¿Cómo obtenemos  $a^{-1} \bmod m$ ?
- Pequeño teorema de Fermat:  $a^{m-1} \equiv 1 \bmod m$ , si  $m$  es primo y  $a$  no es múltiplo de  $m$
- $a^{m-2} \equiv a^{-1} \bmod m$

- **División:**  $(a/b)\%m = (a\%m \cdot b^{-1}\%m)\%m$
- No podemos repartir en la división
- Ej:  $\frac{21}{7} \bmod 5 \neq \frac{1}{2}$
  
- $a^{-1} \bmod m$  es el inverso modular de  $a$  módulo  $m$
- ¿Cómo obtenemos  $a^{-1} \bmod m$ ?
- Pequeño teorema de Fermat:  $a^{m-1} \equiv 1 \bmod m$ , si  $m$  es primo y  $a$  no es múltiplo de  $m$
- $a^{m-2} \equiv a^{-1} \bmod m$
- $(a/b)\%m = (a\%m \cdot b^{m-2}\%m)\%m$

$$(a + b) \% m = (a \% m + b \% m) \% m$$

$$(a - b) \% m = (a \% m - b \% m + m) \% m$$

$$(a \cdot b) \% m = (a \% m \cdot b \% m) \% m$$

$$(a/b) \% m = (a \% m \cdot b^{m-2} \% m) \% m$$



- Para calcular  $b^{m-2} \bmod m$ , necesitamos una forma eficiente de calcular potencias
- Multiplicar  $b$   $m$  veces es  $O(m)$ ,  $m$  suele ser un valor cercano a  $10^9$

- Para calcular  $b^{m-2} \bmod m$ , necesitamos una forma eficiente de calcular potencias
- Multiplicar  $b$   $m$  veces es  $O(m)$ ,  $m$  suele ser un valor cercano a  $10^9$
- La función `pow(b, e)` de C++ encuentra una aproximación, problemas de precisión

# Potenciación binaria

- Para calcular  $b^{m-2} \bmod m$ , necesitamos una forma eficiente de calcular potencias
- Multiplicar  $b$   $m$  veces es  $O(m)$ ,  $m$  suele ser un valor cercano a  $10^9$
- La función `pow(b, e)` de C++ encuentra una aproximación, problemas de precisión
- La función `pow(b, e, m)` de Python es eficiente y precisa. Usa potenciación binaria

- $b^e = b \cdot b \cdot b \cdots b \cdot b \cdot b$

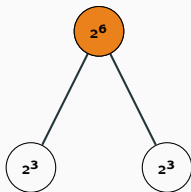
- $b^e = b \cdot b \cdot b \cdots b \cdot b \cdot b$
- Si  $e$  es par  $\rightarrow b^e = b^{e/2} \cdot b^{e/2}$

- $b^e = b \cdot b \cdot b \cdots b \cdot b \cdot b$
- Si  $e$  es par  $\rightarrow b^e = b^{e/2} \cdot b^{e/2}$
- Si  $e$  es impar  $\rightarrow b^e = b^{e-1} \cdot b$

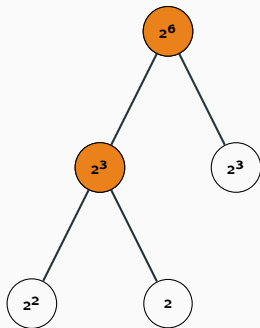
- $b^e = b \cdot b \cdot b \cdots b \cdot b \cdot b$
- Si  $e$  es par  $\rightarrow b^e = b^{e/2} \cdot b^{e/2}$
- Si  $e$  es impar  $\rightarrow b^e = b^{e-1} \cdot b$
- Seguimos reduciendo  $e$  hasta llegar a  $b^0 = 1$  o  $b^1 = b$

$$2^6$$

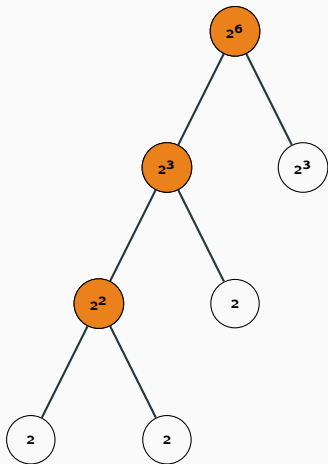




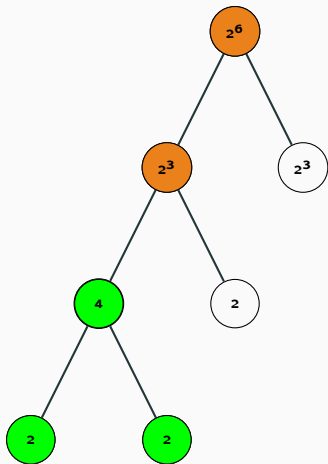
# Potenciación binaria



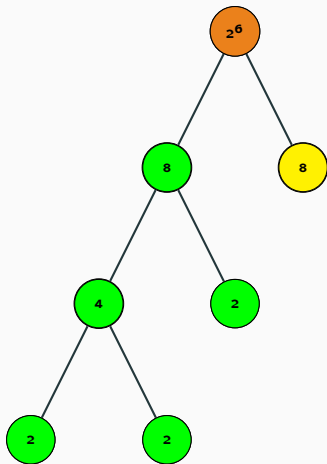
# Potenciación binaria

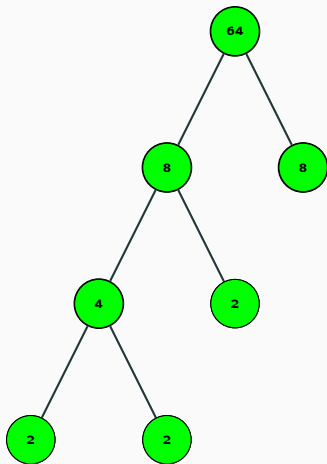


# Potenciación binaria



# Potenciación binaria





# Potenciación binaria

```
int binPow(int b, int e, int mod) {  
    if (e == 0) return 1;  
    if (e == 1) return b;  
    int res;  
    if (e % 2 == 0) {  
        res = binPow(b, e / 2);  
        res = res * res % mod;  
    }  
    else {  
        res = binPow(b, e - 1) * b % mod;  
    }  
    return res;  
}
```