Matemática Básica

Julián Braier

Prim Floyd (FCEN-UBA)

Training Camp Argentino 2021

Contenidos

Primos

Primos

- Decimos que a es un factor o divisor de b si a divide a b. Si a es factor de b notamos $a \mid b$, si no $a \nmid b$.

Primos

- Decimos que a es un factor o divisor de b si a divide a b. Si a es factor de b notamos $a \mid b$, si no $a \nmid b$.
- Un entero $n > 1$ es primo si sus únicos factores positivos son 1 y n.

Primos

- Decimos que a es un factor o divisor de b si a divide a b. Si a es factor de b notamos $a \mid b$, si no $a \nmid b$.
- Un entero $n > 1$ es primo si sus únicos factores positivos son 1 y n.
- Cómo podemos chequear si un número es o no primo?

Primos

- Decimos que a es un factor o divisor de b si a divide a b. Si a es factor de b notamos $a \mid b$, si no $a \nmid b$.
- Un entero $n > 1$ es primo si sus únicos factores positivos son 1 y n.
- Cómo podemos chequear si un número es o no primo?
  - Para cada entero x entre 2 y $\sqrt{n}$ ver si x divide a n. Complejidad: $O(\sqrt{n})$.

Factorización

- Para cada entero $n > 1$ existe una única factorización en primos.

[1] https://cses.fi/problemset/task/1713

Factorización

- Para cada entero $n > 1$ existe una única factorización en primos.

$$n = p_1^{\alpha_1} p_2^{\alpha_2} ... p_k^{\alpha_k},$$

donde $p_1, p_2, ..., p_k$ son primos diferentes y $\alpha_1, \alpha_2, ..., \alpha_k$ son enteros positivos.

[1] `https://cses.fi/problemset/task/1713`

Factorización

- Para cada entero $n > 1$ existe una única factorización en primos.

$$n = p_1^{\alpha_1} p_2^{\alpha_2} ... p_k^{\alpha_k},$$

donde $p_1, p_2, ..., p_k$ son primos diferentes y $\alpha_1, \alpha_2, ..., \alpha_k$ son enteros positivos.

- Por ejemplo, la factorización de 12:

$$12 = 2^2 * 3^1.$$

[1] `https://cses.fi/problemset/task/1713`

Factorización

- Para cada entero $n > 1$ existe una única factorización en primos.

$$n = p_1^{\alpha_1} p_2^{\alpha_2} ... p_k^{\alpha_k},$$

donde $p_1, p_2, ..., p_k$ son primos diferentes y $\alpha_1, \alpha_2, ..., \alpha_k$ son enteros positivos.

- Por ejemplo, la factorización de 12:

$$12 = 2^2 * 3^1.$$

- Llamemos $\tau(n)$ a la cantidad de divisores[1] de n. Ejemplo $\tau(12) = 6$, porque los divisores de 12 son: 1, 2, 3, 4, 6 y 12. Tenemos la fórmula:

$$\tau(n) = \prod_{i=1}^{k} (\alpha_i + 1)$$

[1] `https://cses.fi/problemset/task/1713`

Factorización

- Cómo podemos obtener la factorización de un número?

Factorización

- Cómo podemos obtener la factorización de un número?

```
vector<int> factors(int n) {
   vector<int> f;
   for (int x = 2; x*x <= n; x++) {
      while (n%x == 0) {
         f.push_back(x);
         n /= x;
      }
   }
   if (n > 1) f.push_back(n);
   return f;
}
```

Factorización

- Cómo podemos obtener la factorización de un número?

```
vector<int> factors(int n) {
   vector<int> f;
   for (int x = 2; x*x <= n; x++) {
      while (n%x == 0) {
         f.push_back(x);
         n /= x;
      }
   }
   if (n > 1) f.push_back(n);
   return f;
}
```

- En f los factores aparecerán tantas veces como dividan a n. Ejemplo factors(12) $\rightarrow \{2, 2, 3\}$.

Factorización

- Cómo podemos obtener la factorización de un número?

```
vector<int> factors(int n) {
   vector<int> f;
   for (int x = 2; x*x <= n; x++) {
      while (n%x == 0) {
         f.push_back(x);
         n /= x;
      }
   }
   if (n > 1) f.push_back(n);
   return f;
}
```

- En f los factores aparecerán tantas veces como dividan a n. Ejemplo factors$(12) \rightarrow \{2, 2, 3\}$.
- Complejidad: $O(\sqrt{n})$.

Tests de Primalidad

- Existen algoritmos probabilísticos (y eficientes) para chequear si un número es primo o para factorizar un entero.

[2] https://cp-algorithms.com/algebra/primality_tests.html#toc-tgt-2

[3] https://cp-algorithms.com/algebra/factorization.html#toc-tgt-5

Tests de Primalidad

- Existen algoritmos probabilísticos (y eficientes) para chequear si un número es primo o para factorizar un entero.
- Para testear si un número es primo: Miller Rabin[2].

[2] https://cp-algorithms.com/algebra/primality_tests.html#toc-tgt-2

[3] https://cp-algorithms.com/algebra/factorization.html#toc-tgt-5

Tests de Primalidad

- Existen algoritmos probabilísticos (y eficientes) para chequear si un número es primo o para factorizar un entero.
- Para testear si un número es primo: Miller Rabin[2].
- Para factorizar un número: Pollard's Rho[3].

[2] https://cp-algorithms.com/algebra/primality_tests.html#toc-tgt-2

[3] https://cp-algorithms.com/algebra/factorization.html#toc-tgt-5

Tests de Primalidad

- Existen algoritmos probabilísticos (y eficientes) para chequear si un número es primo o para factorizar un entero.
- Para testear si un número es primo: Miller Rabin[2].
- Para factorizar un número: Pollard's Rho[3].

[2] https://cp-algorithms.com/algebra/primality_tests.html#toc-tgt-2

[3] https://cp-algorithms.com/algebra/factorization.html#toc-tgt-5

Muchos Primos

- Hay infinitos números primos (la demostración es sencilla pero no la quiero hacer).

Muchos Primos

- Hay infinitos números primos (la demostración es sencilla pero no la quiero hacer).
- La "función-contadora-de-primos" $\pi(n)$ nos dice cuántos primos hay $\leq n$. Ejemplo: $\pi(10) = 4$, porque los primos hasta 10 son 2, 3, 5 y 7.

Muchos Primos

- Hay infinitos números primos (la demostración es sencilla pero no la quiero hacer).
- La "función-contadora-de-primos" $\pi(n)$ nos dice cuántos primos hay $\leq n$. Ejemplo: $\pi(10) = 4$, porque los primos hasta 10 son 2, 3, 5 y 7.
- Se puede ver que:

$$\pi(n) \approx \frac{n}{ln\ n}$$

o sea que los primos están cada vez más espaciados, pero son bastante frecuentes (desde 2 hasta 10^9 la máxima distancia entre dos primos consecutivos es 282).

Muchos Primos

- Hay infinitos números primos (la demostración es sencilla pero no la quiero hacer).
- La "función-contadora-de-primos" $\pi(n)$ nos dice cuántos primos hay $\leq n$. Ejemplo: $\pi(10) = 4$, porque los primos hasta 10 son 2, 3, 5 y 7.
- Se puede ver que:

  $$\pi(n) \approx \frac{n}{ln\ n}$$

  o sea que los primos están cada vez más espaciados, pero son bastante frecuentes (desde 2 hasta 10^9 la máxima distancia entre dos primos consecutivos es 282).
- Para qué quería saber eso? Ejemplo: `https://codingcompetitions.withgoogle.com/kickstart/round/0000000000435a5b/000000000077a8e6#problem`

Eratóstenes

Criba de Eratóstenes

- Ponele que queremos saber para cada entero x entre 2 y n si x es primo o no.

Criba de Eratóstenes

- Ponele que queremos saber para cada entero x entre 2 y n si x es primo o no.
- Queremos que si x es primo entonces sieve$[x] = 0$, si no sieve$[x] = 1$.

Criba de Eratóstenes

- Ponele que queremos saber para cada entero x entre 2 y n si x es primo o no.
- Queremos que si x es primo entonces sieve$[x] = 0$, si no sieve$[x] = 1$.
- Solución: inicializamos todos los elementos de sieve en 0. Luego...

```
for (int x = 2; x <= n; x++) {
   if (sieve[x]) continue;
   for (int u = 2*x; u <= n; u += x) {
       sieve[u] = 1;
   }
}
```

Criba de Eratóstenes

- Ponele que queremos saber para cada entero x entre 2 y n si x es primo o no.
- Queremos que si x es primo entonces sieve$[x] = 0$, si no sieve$[x] = 1$.
- Solución: inicializamos todos los elementos de sieve en 0. Luego...

```
for (int x = 2; x <= n; x++) {
   if (sieve[x]) continue;
   for (int u = 2*x; u <= n; u += x) {
       sieve[u] = 1;
   }
}
```

- Complejidad: el loop chico se ejecuta a lo sumo $\frac{n}{x}$ veces para cada x. La complejidad total es:

$$\sum_{x=2}^{n}\left(\frac{n}{x}\right) = \frac{n}{2} + \frac{n}{3} + \ldots = O(n\ log\ n).$$

Criba de Eratóstenes

- Ponele que queremos saber para cada entero x entre 2 y n si x es primo o no.
- Queremos que si x es primo entonces sieve$[x] = 0$, si no sieve$[x] = 1$.
- Solución: inicializamos todos los elementos de sieve en 0. Luego...

```
for (int x = 2; x <= n; x++) {
   if (sieve[x]) continue;
   for (int u = 2*x; u <= n; u += x) {
       sieve[u] = 1;
   }
}
```

- Complejidad: el loop chico se ejecuta a lo sumo $\frac{n}{x}$ veces para cada x. La complejidad total es:

$$\sum_{x=2}^{n}\left(\frac{n}{x}\right) = \frac{n}{2} + \frac{n}{3} + \ldots = O(n\ log\ n).$$

En realidad se puede probar que es $O(n\ log\ log\ n)$, o sea casi $O(n)$.

Criba de Eratóstenes Extendida

- Con una pequeña modificación al algoritmo anterior podemos calcular, para cada x, cuál es el primo k más chico que lo divide.

Criba de Eratóstenes Extendida

- Con una pequeña modificación al algoritmo anterior podemos calcular, para cada x, cuál es el primo k más chico que lo divide.

2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
2	3	2	5	2	7	2	3	2	11	2	13	2	3	2	17	2	19	2

- **Fig. 11.2** An extended sieve of Eratosthenes that contains the smallest prime factor of each number

Criba de Eratóstenes Extendida

- Con una pequeña modificación al algoritmo anterior podemos calcular, para cada x, cuál es el primo k más chico que lo divide.

2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
2	3	2	5	2	7	2	3	2	11	2	13	2	3	2	17	2	19	2

- **Fig. 11.2** An extended sieve of Eratosthenes that contains the smallest prime factor of each number

- De esta manera podemos, además de saber si $x \leq n$ es primo o no en $O(1)$, encotrar su factorización en $O(log\ x)$.

Euclides

Algoritmo de Euclides

- Llamamos máximo común divisor entre a y b ($gcd(a, b)$) al máximo entero que es divisor de a y de b a la vez.

Algoritmo de Euclides

- Llamamos máximo común divisor entre a y b ($gcd(a, b)$) al máximo entero que es divisor de a y de b a la vez.
- Con esta formulación podemos calcular $gcd(a, b)$ eficientemente (con complejidad $O(log\ min(a, b))$).
  - $gcd(a, 0) = a$,
  - $gcd(a, b) = gcd(b, a\ mod\ b), si\ b \neq 0$.

Algoritmo de Euclides

- Llamamos máximo común divisor entre a y b ($gcd(a, b)$) al máximo entero que es divisor de a y de b a la vez.
- Con esta formulación podemos calcular $gcd(a, b)$ eficientemente (con complejidad $O(log\ min(a, b))$).
  - $gcd(a, 0) = a$,
  - $gcd(a, b) = gcd(b, a\ mod\ b), si\ b \neq 0$.
- Llamamos mínimo común múltiplo ($lcm(a, b)$) al entero más chico que es múltiplo de ambos.

$$lcm(a, b) = \frac{a\ b}{gcd(a,b)}$$

Algoritmo de Euclides Extendido

- Podemos extender el algoritmo para que devuelva enteros x e y tales que:

$$ax + by = gcd(a, b)$$

Algoritmo de Euclides Extendido

- Podemos extender el algoritmo para que devuelva enteros x e y tales que:

$$ax + by = gcd(a, b)$$

- Podemos resolver recursivamente de manera parecida a cuando sólo queríamos el gcd. Supongamos que tenemos x', y' tales que:

$$bx' + (a \ mod \ b)y' = gcd(a, b).$$

Algoritmo de Euclides Extendido

- Podemos extender el algoritmo para que devuelva enteros x e y tales que:

$$ax + by = gcd(a, b)$$

- Podemos resolver recursivamente de manera parecida a cuando sólo queríamos el gcd. Supongamos que tenemos x', y' tales que:

$$bx' + (a \ mod \ b)y' = gcd(a, b).$$

- Dado que $(a \ mod \ b) = a - \lfloor \frac{a}{b} \rfloor b$:

$$bx' + (a - \lfloor \frac{a}{b} \rfloor b)y' = gcd(a, b).$$

Algoritmo de Euclides Extendido

- Podemos extender el algoritmo para que devuelva enteros x e y tales que:

$$ax + by = gcd(a, b)$$

- Podemos resolver recursivamente de manera parecida a cuando sólo queríamos el *gcd*. Supongamos que tenemos x', y' tales que:

$$bx' + (a \ mod \ b)y' = gcd(a, b).$$

- Dado que $(a \ mod \ b) = a - \lfloor \frac{a}{b} \rfloor b$:

$$bx' + (a - \lfloor \frac{a}{b} \rfloor b)y' = gcd(a, b).$$

- Moviendo las cosas un poquito...

$$ay' + b(x' - \lfloor \frac{a}{b} \rfloor y') = gcd(a, b).$$

Algoritmo de Euclides Extendido implementación

```
tuple<int,int,int> gcd(int a, int b) {
   if (b == 0) {
       return {1,0,a};
   } else {
       int x,y,g;
       tie(x,y,g) = gcd(b,a%b);
       return {y,x-(a/b)*y,g};
   }
}
```

Algoritmo de Euclides Extendido implementación

```
tuple<int,int,int> gcd(int a, int b) {
   if (b == 0) {
       return {1,0,a};
   } else {
       int x,y,g;
       tie(x,y,g) = gcd(b,a%b);
       return {y,x-(a/b)*y,g};
   }
}
```

- Tiene la misma complejidad que el algoritmo de Euclides en su forma sencilla: $O(log\ min(a, b))$.

Infinitas soluciones

- El algoritmo que les mostré da una sola solución (x, y), pero hay infinitas.

Infinitas soluciones

- El algoritmo que les mostré da una sola solución (x, y), pero hay infinitas.
- Si (x, y) es solución a $ax + by = gcd(a, b)$, entonces para todo entero k

Infinitas soluciones

- El algoritmo que les mostré da una sola solución (x, y), pero hay infinitas.
- Si (x, y) es solución a $ax + by = gcd(a, b)$, entonces para todo entero k

$$(x + \frac{kb}{gcd(a,b)}, y - \frac{ka}{gcd(a,b)})$$

también es solución.

Ecuaciones Diofánticas

- Una ecuación diofántica es una ecuación con la pinta:

$$ax + by = c,$$

donde a, b y c son constantes enteras dadas, y tenemos que hallar x e y que satisfagan. Todos los números deben ser enteros.

Ecuaciones Diofánticas

- Una ecuación diofántica es una ecuación con la pinta:

$$ax + by = c,$$

donde a, b y c son constantes enteras dadas, y tenemos que hallar x e y que satisfagan. Todos los números deben ser enteros.

- Se puede probar que existe solución si y sólo si $gcd(a, b)|c$.
- La solución se encuentra usando euclides extendido para a y b, obteniendo x', y' y $gcd(a, b)$. Una solución a la diofántica es el par (x', y') multiplicado por $\frac{c}{gcd(a,b)}$.

Ecuaciones Diofánticas

- Una ecuación diofántica es una ecuación con la pinta:

$$ax + by = c,$$

donde a, b y c son constantes enteras dadas, y tenemos que hallar x e y que satisfagan. Todos los números deben ser enteros.

- Se puede probar que existe solución si y sólo si $gcd(a, b)|c$.
- La solución se encuentra usando euclides extendido para a y b, obteniendo x', y' y $gcd(a, b)$. Una solución a la diofántica es el par (x', y') multiplicado por $\frac{c}{gcd(a,b)}$.
- Usando las infinitas soluciones para euclides podemos obtener las infinitas soluciones para la diofántica.

Since number of ways can be large, Shaun would report the answer as modulo 1000000007 ($10^9 + 7$).

Figure: Codechef

Since number of ways can be large, Shaun would report the answer as modulo 1000000007 ($10^9 + 7$).

Figure: Codechef

Output

For each test case, print the number of excellent arrays modulo $10^9 + 7$.

Figure: Codeforces

Aritmética Modular

Since number of ways can be large, Shaun would report the answer as modulo 1000000007 ($10^9 + 7$).

Figure: Codechef

Output

For each test case, print the number of excellent arrays modulo $10^9 + 7$.

Figure: Codeforces

parameters and the fixed board contents, please determine the sum of the scores of all those games. Since the output can be a really big number, we only ask you to output the remainder of dividing the result by the prime $10^9 + 7$ (1000000007).

Figure: Codejam

Aritmética Modular

- Decimos que dos enteros a y b son congruentes módulo m si $m \mid b - a$.
- **Notación:** $a \equiv b \ (mod \ m)$.

Aritmética Modular

- Decimos que dos enteros a y b son congruentes módulo m si $m \mid b - a$.
- **Notación:** $a \equiv b \ (mod \ m)$.
- El resto de a módulo m es el único r tal que:
  - $0 \leq r \leq m - 1$ y
  - $a \equiv r \ (mod \ m)$.

Aritmética Modular

- Decimos que dos enteros a y b son congruentes módulo m si $m \mid b - a$.
- **Notación:** $a \equiv b \ (mod \ m)$.
- El resto de a módulo m es el único r tal que:
  - $0 \leq r \leq m - 1$ y
  - $a \equiv r \ (mod \ m)$.
- Detallecito de implementación: el operador % en c++ entre 2 enteros te da el resto del primero módulo el segundo.
  Ejemplo: 8%3 da 2.

Aritmética Modular

- Decimos que dos enteros a y b son congruentes módulo m si $m \mid b - a$.
- **Notación:** $a \equiv b \ (mod \ m)$.
- El resto de a módulo m es el único r tal que:
  - $0 \leq r \leq m - 1$ y
  - $a \equiv r \ (mod \ m)$.
- Detallecito de implementación: el operador % en c++ entre 2 enteros te da el resto del primero módulo el segundo.
  Ejemplo: 8%3 da 2.
  - Con números negativos no funciona como nos gustaría.
    Ejemplo: (-4)%3 da -1, no da 2.

Propiedades Bonitas

- Propiedades:
  - $a + b \equiv r \ (mod \ m) \Rightarrow a\%m + b\%m \equiv r \ (mod \ m)$.
  - $a - b \equiv r \ (mod \ m) \Rightarrow a\%m - b\%m \equiv r \ (mod \ m)$.
  - $a * b \equiv r \ (mod \ m) \Rightarrow a\%m * b\%m \equiv r \ (mod \ m)$.

Propiedades Bonitas

- Propiedades:
  - $a + b \equiv r \ (mod \ m) \Rightarrow a\%m + b\%m \equiv r \ (mod \ m)$.
  - $a - b \equiv r \ (mod \ m) \Rightarrow a\%m - b\%m \equiv r \ (mod \ m)$.
  - $a * b \equiv r \ (mod \ m) \Rightarrow a\%m * b\%m \equiv r \ (mod \ m)$.
- Estas propiedades son bonitas porque nos permiten ir tomando módulo r en pasos intermedios.

Propiedades Bonitas

- Propiedades:
  - $a + b \equiv r \ (mod \ m) \Rightarrow a\%m + b\%m \equiv r \ (mod \ m)$.
  - $a - b \equiv r \ (mod \ m) \Rightarrow a\%m - b\%m \equiv r \ (mod \ m)$.
  - $a * b \equiv r \ (mod \ m) \Rightarrow a\%m * b\%m \equiv r \ (mod \ m)$.
- Estas propiedades son bonitas porque nos permiten ir tomando módulo r en pasos intermedios.
- ¿Vale que $\frac{a}{b} \equiv r \ (mod \ m) \Rightarrow \frac{a\%m}{b\%m} \equiv r \ (mod \ m)$?

Propiedades Bonitas

- Propiedades:
  - $a + b \equiv r \ (mod \ m) \Rightarrow a\%m + b\%m \equiv r \ (mod \ m)$.
  - $a - b \equiv r \ (mod \ m) \Rightarrow a\%m - b\%m \equiv r \ (mod \ m)$.
  - $a * b \equiv r \ (mod \ m) \Rightarrow a\%m * b\%m \equiv r \ (mod \ m)$.
- Estas propiedades son bonitas porque nos permiten ir tomando módulo r en pasos intermedios.
- ¿Vale que $\frac{a}{b} \equiv r \ (mod \ m) \Rightarrow \frac{a\%m}{b\%m} \equiv r \ (mod \ m)$?
- Lamentablemente no. Contraejemplo: con $a = 6, b = 2, m = 4$...
  - $\frac{a}{b} \equiv \frac{6}{2} \equiv 3 \ (mod \ m)$, pero
  - $\frac{a\%m}{b\%m} \equiv \frac{6\%4}{2\%4} \equiv 1 \ (mod \ m)$

Inverso Modular

- No podemos dividir... pero podemos multiplicar por el inverso modular.

Inverso Modular

- No podemos dividir... pero podemos multiplicar por el inverso modular.
- Dados $x \not\equiv 0$ y p primo, llamamos inverso modular de x (x^{-1}) al entero tal que $x * x^{-1} \equiv 1 \ (mod \ m)$.

Inverso Modular

- No podemos dividir... pero podemos multiplicar por el inverso modular.
- Dados $x \not\equiv 0$ y p primo, llamamos inverso modular de x (x^{-1}) al entero tal que $x * x^{-1} \equiv 1 \ (mod \ m)$.
- Para calcular $\frac{a}{b}$ % m lo que hacemos es calcular $a * b^{-1}$ % m.

Inverso Modular

- No podemos dividir... pero podemos multiplicar por el inverso modular.
- Dados $x \not\equiv 0$ y p primo, llamamos inverso modular de x (x^{-1}) al entero tal que $x * x^{-1} \equiv 1 \ (mod \ m)$.
- Para calcular $\frac{a}{b}$ % m lo que hacemos es calcular $a * b^{-1}$ % m.
- ¿Y cómo calculamos el inverso?

Fermat

Inverso Modular

Pequeño Teorema de Fermat

Dado $x \not\equiv 0$ y p primo, $x^{p-1} \equiv 1 \ (mod \ p)$.

[4] https://cp-algorithms.com/algebra/module-inverse.html#mod-inv-all-num

Inverso Modular

Pequeño Teorema de Fermat

Dado $x \not\equiv 0$ y p primo, $x^{p-1} \equiv 1 \ (mod \ p)$.

- Entonces $x^{p-1} \equiv x * x^{p-2} \equiv 1 \ (mod \ p)$. O sea x^{p-2} es el inverso de x módulo p.

[4] https://cp-algorithms.com/algebra/module-inverse.html#mod-inv-all-num

Inverso Modular

Pequeño Teorema de Fermat

Dado $x \not\equiv 0$ y p primo, $x^{p-1} \equiv 1 \ (mod \ p)$.

- Entonces $x^{p-1} \equiv x * x^{p-2} \equiv 1 \ (mod \ p)$. O sea x^{p-2} es el inverso de x módulo p.
- ¿Y ahora? ¿Cómo calculamos x^{p-2}?

[4] https://cp-algorithms.com/algebra/module-inverse.html#mod-inv-all-num

Inverso Modular

Pequeño Teorema de Fermat

Dado $x \not\equiv 0$ y p primo, $x^{p-1} \equiv 1 \ (mod \ p)$.

- Entonces $x^{p-1} \equiv x * x^{p-2} \equiv 1 \ (mod \ p)$. O sea x^{p-2} es el inverso de x módulo p.
- ¿Y ahora? ¿Cómo calculamos x^{p-2}?
- Calcular $x * x * ... * x$ $p-2$ veces es caro. Pero podemos calcular x^{p-2} en $O(log \ p)$ con la siguiente recurrencia:
  - Si n es 0: x^n es 1,
  - Si $n > 0$ y n par: x^n es $(x^{\frac{n}{2}})^2$,
  - Si $n > 0$ y n impar: x^n es $(x^{\frac{n-1}{2}})^2 * x$.

[4] `https://cp-algorithms.com/algebra/module-inverse.html#mod-inv-all-num`

Inverso Modular

Pequeño Teorema de Fermat

Dado $x \not\equiv 0$ y p primo, $x^{p-1} \equiv 1 \ (mod \ p)$.

- Entonces $x^{p-1} \equiv x * x^{p-2} \equiv 1 \ (mod \ p)$. O sea x^{p-2} es el inverso de x módulo p.
- ¿Y ahora? ¿Cómo calculamos x^{p-2}?
- Calcular $x * x * ... * x$ $p-2$ veces es caro. Pero podemos calcular x^{p-2} en $O(log \ p)$ con la siguiente recurrencia:
  - Si n es 0: x^n es 1,
  - Si $n > 0$ y n par: x^n es $(x^{\frac{n}{2}})^2$,
  - Si $n > 0$ y n impar: x^n es $(x^{\frac{n-1}{2}})^2 * x$.
- De esta manera podemos dividir módulo p en $O(log \ p)$. Esto suele ser suficientemente eficiente. Si no existe una manera de precomputar en tiempo lineal a todos los inversos módulo p.[4]

[4] `https://cp-algorithms.com/algebra/module-inverse.html#mod-inv-all-num`

Combinatoria

- En combinatoria se estudian métodos para contar combinaciones de objetos.

Combinatoria

- En combinatoria se estudian métodos para contar combinaciones de objetos.
- Usualmente, el objetivo es encontrar una manera eficiente de contar las combinaciones, sin generar cada una de las combinaciones por separado.

Coeficientes Binomiales

- El coeficiente binomial (a.k.a número combinatorio) $\binom{n}{k}$ nos dice cuántas maneras hay de tomar un subconjunto de k elementos de un conjunto de n elementos.

Coeficientes Binomiales

- El coeficiente binomial (a.k.a número combinatorio) $\binom{n}{k}$ nos dice cuántas maneras hay de tomar un subconjunto de k elementos de un conjunto de n elementos.
- Se pueden definir recursivamente con la fórmula:
  - $\binom{n}{0} = \binom{n}{n} = 1$,
  - $\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$.

Coeficientes Binomiales

- El coeficiente binomial (a.k.a número combinatorio) $\binom{n}{k}$ nos dice cuántas maneras hay de tomar un subconjunto de k elementos de un conjunto de n elementos.
- Se pueden definir recursivamente con la fórmula:
  - $\binom{n}{0} = \binom{n}{n} = 1$,
  - $\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$.
- Maneras alternativas de computar los combinatorios:
  1. $\binom{n}{k} = \frac{n!}{k!(n-k)!}$.
  2. $\binom{n}{k} = \frac{n}{k}\binom{n-1}{k-1}$ (sólo si $k \neq 0$).

Coeficientes Binomiales

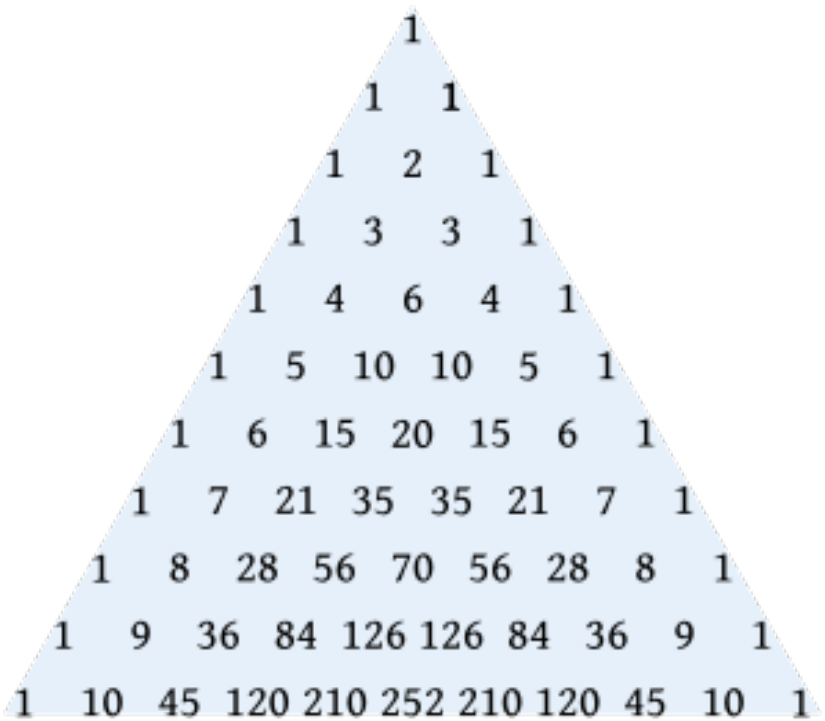
- El coeficiente binomial (a.k.a número combinatorio) $\binom{n}{k}$ nos dice cuántas maneras hay de tomar un subconjunto de k elementos de un conjunto de n elementos.
- Se pueden definir recursivamente con la fórmula:
  - $\binom{n}{0} = \binom{n}{n} = 1$,
  - $\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$.
- Maneras alternativas de computar los combinatorios:
  1. $\binom{n}{k} = \frac{n!}{k!(n-k)!}$.
  2. $\binom{n}{k} = \frac{n}{k}\binom{n-1}{k-1}$ (sólo si $k \neq 0$).
- Ejemplo: si en un torneo de 20 equipos juegan todos contra todos, cuántos partidos tiene el torneo?
  - Respuesta: $\binom{20}{2} = 190$.

Coeficientes Binomiales

Propiedades:

- $\binom{n}{k} = \binom{n}{n-k}$.
- $\sum_{k=0}^{n} \binom{n}{k} = \binom{n}{0} + \binom{n}{1} + \ldots + \binom{n}{n} = 2^n$.

Triángulo de Pascal



El Libro

Me basé bastante en el libro de Antti Laaksonen "Guide to Competitive Programming"[5] (se publicó una reedición en 2020). Recomiendo fuertemente.

[5] https://link.springer.com/book/10.1007/978-3-319-72547-5