Problemas “matemáticos”

Pedro Pablo Gómez Martín

Facultad de Informática - UCM

Renuncia de responsabilidad

- **No** soy matemático.
- Las matemáticas son un área *muy amplia*.
  - Probabilidad
  - Teoría de números
  - Búsqueda de ciclos
  - Números grandes
  - Progresiones
  - Combinatoria
  - Álgebra
  - Teoría de juegos
  - ...
- Solo arañaremos la superficie (más en geometría computacional)
- Muchos problemas requieren análisis matemáticos previos ...
- ...lo que puede convertirlos en *acertijos*
- No son muy habituales en entrevistas de trabajo

Algunas fórmulas

- Suma de una progresión aritmética:

$$1 + 2 + 3 + \ldots + n = \frac{n(n+1)}{2}$$

$$a + (a+d) + (a+2d) + \overset{\text{n términos}}{\ldots} = \frac{n(2a + (n-1)d)}{2}$$

- Suma de una progresión geométrica:

$$1 + 2 + 4 + \ldots + 2^n = 2^{n+1} - 1$$

$$b^0 + b^1 + b^2 + \overset{\text{n términos}}{\ldots} = \frac{1 - b^n}{1 - b}$$

- Coeficiente binomial

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k} = \frac{n!}{k! \cdot (n-k)!}$$

Límite de la representación (64 bits)

Tipo	Tamaño (bytes)	Límite
`char`	1	-128...127
`short`	2	-32.768...32.767
`int`	4	-2.147.483.648...2.147.483.647
`long`	8	±9.223.372.036.854.775.808
`uchar`	1	0...255
`ushort`	2	0...65.537
`uint`	4	0...4.294.967.295
`ulong`	8	0...18.446.744.073.709.551.615

Límite de la representación (64 bits)

En plataformas de 32 bits, `long` *es sinónimo* de `int`. Hay que usar `[unsigned] long long` para tener 8 bytes.

`http://pcom.fdi.ucm.es` es de 64 bits pero otros jueces podrían ser de 32.

Apuesta sobre seguro: `int` y `long long`.

Límite de la representación (64 bits)

Tipo	Límite
`int`	$\approx 2 \cdot 10^9 < 13!$
`long`	$\approx 9 \cdot 10^{18} < 21!$

Si no tienes suficiente, aritmética de precisión arbitraria

Números grandes

- Los números naturales son secuencias de dígitos que podemos guardar como una cadena
- Podemos programar las operaciones aritméticas procesando cadenas

Números grandes

```
string suma(string a, string b) {
  string r;
  if (a.length() < b.length()) a.swap(b);
  b = string(a.length() - b.length(), '0') + b;

  unsigned int i = a.length(), carry = 0;
  while(i--) {
    carry += a[i] + b[i] - 2*'0';
    r.push_back('0' + carry % 10);
    carry /= 10;
  }
  if (carry)
    r.push_back(carry + '0');
  reverse(r.begin(), r.end());

  return r;
}
```

Números grandes

- La resta es más delicada (si necesitamos soportar negativos)
- La multiplicación es más laboriosa. La implementación obvia puede ser lenta. Karatsuba
- Con la división y el módulo el asunto se pone emocionante
  - Más fácil si solo necesitamos `BigNumber <op> int`.
- Para la raíz cuadrada ya hay que remangarse (UVa 10023 Square root)

Consejo

Si te enfrentas a un problema con números grandes, quizá haya llegado el momento de usar Java: `BigInteger`

. . . pero no siempre te hará falta.

Números primos

- Un número es primo si solo es divisible por 1 y por él mismo: 2, 3, 5, 7, ...
- Para saber si un número es primo, probamos a dividir por todos los números empezando en el 2
  - ¡¡Basta con llegar a $\sqrt{n}$!! $\mathcal{O}(\sqrt{n}) \ll \mathcal{O}(n)$

Números primos

- Si queremos sacar los factores primos hay que *ir dividiendo* y mirar *el residuo final*.
- Es un problema de primero :-)

```
void factoriza(unsigned int i) {
  assert(i >= 2); // O comprobaciones especiales
  unsigned int p = 2;
  while(p*p <= i) { // ¡Mucho mejor que sqrt!
    while(!(i % p)) {
      cout << p << ' ';
      i /= p;
    }
    ++p;
  }
  if (i != 1) // Podría quedarnos un único primo
    cout << i << ' ';
  cout << '\n';
}
```

Números primos

- Podemos probar primero con 2 y luego pasar solo por los impares
- Se puede adaptar para muchas otras cosas:
  - ¿Cuántos factores primos tiene un número?
  - ¿Cuántos divisores tiene un número? (multiplicación de los exponentes más uno)
  - ¿Cuál es la suma de los divisores de un número?

Números primos

- Si tenemos que hacer muchos test de primalidad es mejor precalcular los primos
  - Cuidado con los tamaños. Normalmente hasta $\sqrt{MAX\_N}$
- Vamos metiendo los primos en un vector
- Para los números siguientes probamos a dividir solo por los primos encontrados
- Aproximación de Gauss:

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

- Hay aproximadamente 48.254.942 primos menores que 10^9 (en realidad, algo más de 50 millones)

Si `MAX_N` es grande, *es lento*. Hay muchos números primos, y nos hacen llegar hasta muy arriba.

Criba de Eratóstenes

- Array de booleanos que nos dice si un número es o no primo.
- Al principio todos a cierto.
- Vamos “tirando” los múltiplos de los números primos que nos encontramos.

```
bitset<MAX_PRIME+1> bs; // #include <bitset>
vector<uint> primes;    // unsigned int
void sieve() {
  bs.set(); // De momento, todos son primos
  bs[0] = bs[1] = 0; // El 0 y el 1 no lo son.
  for (uint i = 2; i <= MAX_PRIME; ++i) {
    if (bs[i]) {
      // El actual es primo. Tiramos sus múltiplos
      for (uint j = i*i; j <= MAX_PRIME; j += i)
        bs[j] = 0; // i*i en el for ¡Cuidado con el rango!
      primes.push_back(i);
    } // if es primo
  } // for
} // sieve
```

Criba de Eratóstenes

```
bool isPrime(unsigned long long n) {
  if (n <= MAX_PRIME)
    return bs[n];
  for (unsigned int i = 0;
       primes[i]*primes[i] <= n; ++i) {
    if (!(n % primes[i]))
      return false;
  }
  return true;
}
```

- Es *mucho más rápido*. Calcular los primos hasta 10^7 es viable.
  - Esto nos permite factorizar números hasta 10^{14}
- La idea de recorrer los múltiplos de los primos encontrados se puede adaptar para otras cosas

Problemas propuestos

- 124 ¿Cuántas me llevo?
- 402 Las piezas del puzzle