Problemas “matemáticos” (y 2)

Pedro Pablo Gómez Martín

Facultad de Informática - UCM

25 de noviembre de 2020

Aritmética modular

- Para evitar el uso de números grandes muchos problemas piden el resultado módulo m.
  - El m suele ser un número primo (típicamente 1.000.000.007)

Error de principiante

No hay que calcular el módulo *al final* o se habrá sufrido desbordamiento por el camino.

Aritmética modular

- Aprovechamos la *aritmética del reloj* para "volver" al rango de la representación antes de salirnos.
- Propiedad de la suma (simétricamente resta):

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

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

- Propiedad de la multiplicación:

$$(a \times b) \,\%m = ((a \,\%m) \times (b \,\%m)) \,\%m$$

Cuidado

Al calcular $(a \,\%m) \times (b \,\%m)$ *podrías desbordar*. Con $m = 1{,}000{,}000{,}007$ hay que usar `long long`

Aritmética modular

¡¡Cuidado!!

No se cumple para la división:

$$(a \div b) \,\%m \neq ((a \,\%m) \div (b \,\%m)) \,\%m$$

En los problemas donde se usa, *normalmente* no hay que dividir.

MCD y mcm

- Máximo Común Divisor (MCD, GCD en inglés) de dos números a y b:
  - Mayor número que divide simultáneamente a a y a b.
  - Factores primos comunes al menor exponente.
- Mínimo común múltiplo (mcm, lcm en inglés) de dos números a y b:
  - Menor número que es múltiplo simultáneamente a a y a b.
  - Factores primos comunes y no comunes al mayor exponente.

Algoritmo de Euclides

```
int gcd(int a, int b) {
  return b == 0 ? a : gcd(b, a % b);
}

int lcm(int a, int b) {
  return a * (b / gcd(a, b));
}
```

- Desde C++17 (arriesgado aún), en `algorithm`
  - `std::gcd(a,b)` (`std::__gcd(a,b)` en versiones previas)
  - `std::lcm(a,b)`

Ecuaciones diofánticas

- Una *ecuación diofántica* es una ecuación algebraica [de dos o más incógnitas], en las que [los coeficientes son enteros y] se buscan *soluciones enteras*.

Transeúnte, esta es la **tumba de Diofanto**: los números pueden mostrar, ¡oh maravilla! la **duración de su vida**. Su niñez ocupó la **sexta parte** de su vida; después, durante la **doceava parte**, de vello se cubrieron sus mejillas. Pasó aún una **séptima parte** de su vida antes de tomar esposa y, **cinco años después**, tuvo un precioso niño que, una vez alcanzada **la mitad de la edad de la vida de su padre**, pereció de una muerte desgraciada. Su padre tuvo que sobrevivirle, llorándole, durante **cuatro años**. De todo esto se deduce su edad.

$$\frac{x}{6} + \frac{x}{12} + \frac{x}{7} + 5 + \frac{x}{2} + 4 = x$$

Ecuaciones diofánticas

- *No existe* un método general para resolver ecuaciones diofánticas.
- Nos preocuparemos de las que tienen una forma muy concreta:

$$A \cdot x + B \cdot y = C$$

  - A, B y C son valores conocidos.
  - x, y son incógnitas (enteras).
- Tiene solución sí y solo sí

$$C = k \cdot mcd(A, B)$$

Ecuaciones diofánticas: identidad de Bézout

- Si una ecuación diofántica como la anterior tiene solución, entonces *tiene infinitas soluciones*.
- Si $d = mcd(A, B)$ y (x_0, y_0) es una solución de la ecuación, las demás son de la forma:

$$x = x_0 + \lambda \frac{B}{d}$$

$$y = y_0 - \lambda \frac{A}{d}$$

Ecuaciones diofánticas: Algoritmo de Euclides Extendido

- El *algoritmo de Euclides extendido* es una versión "enriquecida" del algoritmo de Euclides que calcula el máximo común divisor y, como subproducto, proporciona la pareja de números (u, v) tales que:

$$A \cdot u + B \cdot v = mcd(A, B)$$

- Se utiliza para calcular *la primera solución*:
  - Se ejecuta el algoritmo de Euclides extendido sobre los coeficientes de la ecuación.
  - Si el mcd generado no divide a C, no hay solución.
  - Si no, multiplicamos u y v por $C/mcd(A, B)$ para tener la primera solución.
  - Dependiendo del problema, usamos la identidad de Bézout para generar más soluciones.

Cuidado

(u, v) pueden ser grandes y la multiplicación para calcular x_0 y y_0 hacerlos crecer más. Plantearse usar `long long`.

Algoritmo de Euclides Extendido

```
int extendedEuclidRec(int a, int b, int &u, int &v) {

  if (!b) {
    u = 1;
    v = 0;
    return a;
  }

  int r = extendedEuclidRec(b, a % b, u, v);
  int uAux = v;
  int vAux = u - (a / b) * v;
  u = uAux;
  v = vAux;
  return r;

} // extendedEuclidRec
```

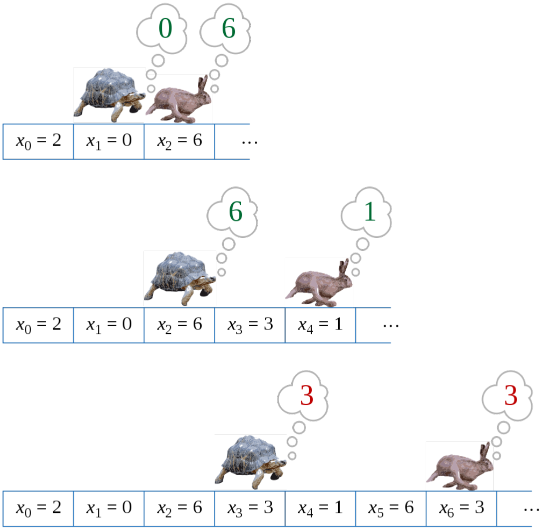
Búsqueda de ciclos

Dada una función $f : S \to S$ (con S de tamaño finito) y un número x_0 de partida, se definen los *iterated function values* a:

$$x_0, x_1 = f(x_0), x_2 = f(x_1), x_3 = f(x_2), \ldots$$

En algún momento se formará un ciclo. ¿En qué punto empieza y cuál es su longitud?

Búsqueda de ciclos: algoritmo de Floyd



Búsqueda de ciclos: algoritmo de Floyd

```
typedef pair<unsigned int, unsigned int> uu;
// <μ,λ> : <inicio ciclo, longitud ciclo>
uu floydCycleFinding(unsigned int x0) {
  int tortoise = f(x0), hare = f(f(x0));
  while (tortoise != hare) {
    tortoise = f(tortoise); hare = f(f(hare));
  }
  int mu = 0; hare = x0;
  while (tortoise != hare) {
    tortoise = f(tortoise); hare = f(hare); mu++;
  }
  int lambda = 1; hare = f(tortoise);
  while (tortoise != hare) {
    hare = f(hare); lambda++;
  }
  return uu(mu, lambda);
}
```

Problemas propuestos

- 399 Las perlas de la condesa
- 287 Las ganancias del hotel