

APÉNDICE. Algoritmos

En este capítulo presentamos los algoritmos del libro en el lenguaje Python. Elegimos Python porque, entre otras cosas, permite definir en una misma instrucción más de una variable. Como ejemplo, si tenemos $a = 2$ y $b = 1$, luego de la instrucción:

```
a,b = a+b,a-b
```

los valores de a y b serán respectivamente 3 (porque $3 = 2+1$) y 1 (porque $1 = 2 - 1$).

□ 1. Algoritmo de división

Éste es el algoritmo de división que vimos en la sección 3.2 del capítulo 2. No es el algoritmo más rápido, porque va restando del número original el divisor, hasta que se queda con un resto pequeño.

```
def division(n,d):
    if d==0:
        return 0,0
    if (n>=0) and (d>0):
        q,r = 0,n
        while (r >= d):
            q,r = q+1,r-d
        return q,r
    if (n>=0) and (d<0):
        q,r = division(n,-d)
        return -q,r
    if (n<0) and (d>0):
        q,r = division(-n,d)
        if r==0:
            return -q,0
        else:
            return -q-1,d-r
    else:
        q,r = division(-n,-d)
        if r==0:
            return q,0
        else:
            return q+1,-r-d
```

En realidad, en Python ya hay una implementación del algoritmo de división. El cociente entre los números enteros a y b se obtiene con $\frac{a}{b}$, mientras que el resto se obtiene con $a\%b$.

□ 2. Escritura en una nueva base

La función `baseb(n,b)` devuelve como secuencia la escritura del número n en base b , siguiendo el algoritmo de la sección 4 del capítulo 2.

```
def baseb(n,b):
    m = n
    s = [ ]
    while m != 0:
        q = m/b
        r = m%b
        s = [r] + s
        m = q
    return s
```

En Python se usa `!=` en lugar de \neq . La instrucción `s = [r] + s` agrega el número r al comienzo de la lista s .

□ 3. Algoritmo de Euclides

La función `mcd(a,b)` devuelve el máximo común divisor entre a y b siguiendo el algoritmo de Euclides de la sección 5.1 del capítulo 2.

```
def mcd(a,b):
    r1,r2 = a,b
    while r2 != 0:
        r = r1 % r2
        r1,r2 = r2,r
    return r1
```

La función `mcdcompleto(a,b)` devuelve tres números: d , s , t , donde

- d es el *mcd* entre a y b
- s y t son enteros que permiten escribir d en términos de a y b : $d = s \cdot a + t \cdot b$.

```
def mcdcompleto(a,b):
    r1,r2 = a,b
    s1,s2,t1,t2 = 1,0,0,1
    while r2 != 0:
        q = r1 / r2
        r = r1 % r2
        s1,s2 = s2,s1-q*s2
        t1,t2 = t2,t1-q*t2
        r1,r2 = r2,r
    return r1,s1,t1
```

El algoritmo funciona correctamente porque en cada paso se tiene que:

- $r_1 = s_1 \cdot a + t_1 \cdot b$, y
- $r_2 = s_2 \cdot a + t_2 \cdot b$.

En cada paso, se reemplazan r_1 y r_2 por r_2 y r , donde r es el resto de dividir r_1 por r_2 , es decir, $r_1 = q \cdot r_2 + r$. Para seguir teniendo las igualdades anteriores, se calcula:

$$\begin{aligned} r &= r_1 - q \cdot r_2 = (s_1 \cdot a + t_1 \cdot b) - q \cdot (s_2 \cdot a + t_2 \cdot b) \\ &= (s_1 - q \cdot s_2) \cdot a + (t_1 - q \cdot t_2) \cdot b \end{aligned}$$

Además, esta igualdad vale en el primer paso, pues $r_1 = a = 1 \cdot a + 0 \cdot b$ y $r_2 = b = 0 \cdot a + 1 \cdot b$. Por lo tanto, vale hasta que se termina el algoritmo. Entonces, como en el último paso el *mcd* está en la variable r_1 , los coeficientes s_1 y t_1 dan la combinación lineal buscada.

□ 4. Ecuaciones diofánticas y de congruencia

El siguiente algoritmo devuelve una solución de la ecuación $a \cdot x + b \cdot y = c$ si $\text{mcd}(a, b) \mid c$. Si no hay solución, devuelve el par `None, None`, que indica que no hay solución posible. Todas las soluciones (y la solución particular encontrada) se calculan como vimos en la sección 1 del capítulo 3.

```
def resolver(a,b,c):
    d = mcd(a,b)
    if (c%d) != 0:
        return None,None
    else:
        r,s,t = mcdcompleto(a,b)
        return s*c/d,t*c/d
```

El algoritmo que sigue busca las soluciones de la ecuación $a \cdot x \equiv b \pmod{m}$. Si no las hay, devuelve `None, None`. Si las hay, devuelve el par $x_0, \frac{m}{\text{mcd}(a,m)}$, mediante el cual se pueden encontrar todas las soluciones como $x \equiv x_0 \pmod{\frac{m}{\text{mcd}(a,m)}}$. Vimos esto en la sección 3 del capítulo 3.

```
def ecuacong(a,b,m):
    d = mcd(a,m)
    if (b%d) != 0:
        return None,None
    a,m,b = a/d, m/d, b/d
    s,t = resolver(a,m,b)
    return s % m, m
```

El siguiente algoritmo es una variante del presentado en la sección 6 del capítulo 3. Si se usa con listas, como en `teochino([14,17],[49,45])`, devuelve las soluciones del sistema:

$$\begin{cases} x \equiv 14 \pmod{49} \\ x \equiv 17 \pmod{45} \end{cases}$$

en la forma (602, 2.205), que significa que las soluciones son los enteros $x \equiv 602 \pmod{2.205}$.

Esta versión también se puede aplicar en el caso en que los módulos m_1, \dots, m_n no son coprimos de a pares. Como en este caso a veces no hay solución, si no la hay el algoritmo devuelve `None, None`.

```
def teochino(a,m):
    M = m[0]
    A = a[0]
    for i in range(len(m)-1):
        Q,n = ecuacong(M,a[i+1]-A,m[i+1])
        if Q==None:
            return None,None
        M,A = M*n,M*Q+A
    return A,M
```

□ 5. Desarrollo decimal de un número racional

Si a y b son números naturales, con el siguiente algoritmo devolvemos (e, Q_1, Q_2) , donde $\frac{a}{b} = e, Q_1\overline{Q_2}$. Por ejemplo, decimal (19,14) devuelve:

(1, [3], [5, 7, 1, 4, 2, 8])

Significa que $\frac{19}{14} = 1,3\overline{571428}$. Entonces, el algoritmo presentado en la sección 3 del capítulo 4, es:

```
def decimal(a,b):
    e = a//b
    r = a%b
    R,Q = [ ],[ ]
    while r != 0:
        if r in R:
            return e,Q[:R.index(r)],Q[R.index(r):]
        R = R + [r]
        Q = Q + [(10*r)//b]
        r = (10*r) % b
    return e,Q,[ ]
```

El condicional `if r in R` es verdadero si el elemento r figura en la lista R . La función `R.index(r)` devuelve la posición en que se encuentra. Por último, `Q[:i]` da la lista Q desde el comienzo hasta la posición anterior a i , mientras que `Q[i:]` devuelve el resto.