El Algoritmo de Euclides

Pablo L. De Nápoli

Departamento de Matemática
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

25 de abril de 2014

Parte I

El algoritmo de Euclides

El algoritmo de Euclides

El algoritmo de Euclides es un algoritmo para el cálculo del máximo común divisor. Sean $a, b \in \mathbb{N}_0$. Para calcular $\mathrm{mcd}(a, b)$ podemos suponer $a \geq b$ (sino intercambiamos los roles pues $\mathrm{mcd}(a, b) = \mathrm{mcd}(b, a)$

Efectuamos entonces la división entera de a por b obteniendo un primer cociente q_1 y un primer resto r_1

$$a = q_1 b + r_1 \quad 0 \leq r_1 < b$$

Si $r_1 = 0$ el algorimo termina. Sino, dividimos a b por r_1 obteniendo un segundo cociente q_2 y un segundo resto r_2

$$b = q_1 r_1 + r_2 \quad 0 \leq r_2 < r_1$$

Si $r_2 = 0$ el algorimo termina. Sino, dividimos a r_1 por r_2 obteniendo un tercer cociente q_3 y un tercer resto r_3

$$r_1 = q_2 r_2 + r_3 \quad 0 \leq r_3 < r_2$$

El algoritmo de Euclides (continuación)

Repitiendo (inductivamente) el procedimiento, supongamos que ya calculamos r_k. Si $r_k = 0$ el algoritmo termina. Si no dividimos a r_{k-1} por r_k, obteniendo que

$$r_{k-1} = q_k r_k + r_{k+1} \quad 0 \leq r_{k+1} < r_k$$

Notamos que por construcción la sucesión de restos

$$r_1 > r_2 > r_3 > \dots$$

es una sucesión de enteros no negativos estrictamente decreciente. Se deduce que esta sucesión no puede continuar indefinidamente. En consecuencia, siempre existe un $N \in \mathbb{N}$ tal que

$$r_N = 0$$

es decir que el algoritmo de Euclides siempre termina.

El algoritmo de Euclides (continuación)

Teorema

El algoritmo de Euclides siempre termina y el último resto no nulo r_{N-1} que se obtiene es el máximo común divisor entre a y b.

Ejemplo: Calculemos el máximo común divisor entre 32 y 17 por el algoritmo de Euclides:

$32 = 1 * 17 + 15$
$17 = 1 * 15 + 2$
$15 = 7 * 2 + 1$
$2 = 2 * 1 + 0$

El algoritmo de Euclides termina: El mcd entre 32 y 17 es 1.

El invariante del Algoritmo de Euclides

Para demostrar el teorema, usamos el siguiente lema que expresa un invariante del algoritmo (una propiedad que se mantiene a lo largo de las iteraciones del algoritmo.

Lema

Si efectuamos la división entera de a por b, obteniendo un cociente q y un resto r, entonces

$$mcd(a, b) = mcd(b, r)$$

La prueba del lema es inmediata: si d es un divisor común entre a y b, entonces d también divide a $r = a - qb$, en consecuencia d es un divisor común entre b y r.

Recíprocamente si d divide a b y r, también dividirá a a pues $a = bq + r$. Se deduce que b y r tienen los mismos divisores comunes que los que tenían a y b, y en consecuencia tienen el mismo máximo común divisor.

Prueba de la correctitud del Algoritmo de Euclides

Usando el lema vemos que la sucesión de restos construida por el algoritmo de Euclides, cumple que

$$\mathrm{mcd}(a, b) = \mathrm{mcd}(b, r_1) = \mathrm{mcd}(r_1, r_2) = \mathrm{mcd}(r_2, r_3) = \ldots$$

$$\ldots = \mathrm{mcd}(r_{N-2}, r_{N-1}) = \mathrm{mcd}(r_{N-1}, 0) = r_{N-1}$$

(pues para cualquier entero $c \in \mathbb{N}_0$, $\mathrm{mcd}(c, 0) = c$)

Es decir que hemos demostrado que el algoritmo de Euclides calcula correctamente el máximo común divisor.

Programita recursivo (en Python 3) para el Algorimo de Euclides

```
def mcd(a,b):
    if b>a:
        return mcd(b,a)
    if b==0:
        print ("El algoritmo de Euclides termina:")
        print ("El mcd es",a)
        return a
    else:
        q,r = divmod(a,b)
        print(a,"=",q,"*",b,"+",r)
        return mcd(b,r)
```

Parte II

Una consecuencia del algoritmo de Euclides: El máximo común divisor se escribe como combinación lineal

El MCD como combinación lineal

Una consecuencia muy importante del algoritmo de Euclides es el siguiente teorema:

Teorema

El máximo común divisor entre a y b se puede escribir como una combinación lineal de ellos: es decir, existen enteros $\alpha = \alpha(a, b)$ *y* $\beta = \beta(a, b)$ *tales que*

$$\alpha a + \beta b = mcd(a, b)$$

El algoritmo de Euclides nos permite dar una prueba constructiva de este teorema, construyendo las funciones $\alpha(a, b)$ y $\beta(b, a)$ de manera recursiva. De nuevo, podemos suponer $a \geq b$.

Definiendo $\alpha(a, b)$ y $\beta(a, b)$ recursivamente

Para encontrar la recurrencia, efectuamos la división entera de a por b, obteniendo un cociente $q = q(a, b)$ y un resto $r = r(a, b)$, como antes. Entonces por el invariante del algoritmo de Euclides sabemos que

$$\text{mcd}(a, b) = \text{mcd}(b, r)$$

Ahora supongamos que sabemos encontrar $\alpha(b, r)$ y $\beta(b, r)$ tales que

$$\alpha(b, r)\ b + \beta(b, r)\ r = \text{mcd}(b, r)$$

Sustituyendo $r = a - bq$, encontramos que

$$\alpha(b, r)\ b + \beta(b, r)\ [a - bq] = \text{mcd}(b, r)$$

y efectuando la distributiva:

$$\beta(b, r)\ a + (\alpha(b, r) - \beta(b, r)q)b = \text{mcd}(b, r)$$

Esto sugiere definir las funciones α y β recursivamente por:

$$\alpha(a, b) = \beta(b, r), \quad \beta(a, b) = \alpha(b, r) - \beta(b, r)q$$

Definiendo $\alpha(a,b)$ y $\beta(a,b)$(continuación)

Para que esta recurrencia funcione, debemos definir α y β cuando el algoritmo de Euclides termina, es decir $\alpha(a,0)$ y $\beta(a,0)$.

Queremos

$$\alpha(a,0)\ a + \beta(a,0)\ 0 = \text{mcd}(a,0) = a$$

Una manera de lograrlo es definiendo

$$\alpha(a,0) = 1$$

$$\beta(a,0) = 0$$

(Esta elección es arbitraria, pero necesaria si quereemos que β sea una función bien definida)

Por inducción global en $a \in \mathbb{N}_0$, se prueba facilmente que $\alpha, \beta : D \to \mathbb{Z}$ quedan bien definidas en el dominio

$$D = \{(a,b) \in \mathbb{N}_0 \times \mathbb{N}_0 : a \geq b\}$$

y que cumplen que:

$$\alpha(a,b)\ a + \beta(a,b)\ b = mcd(a,b)$$

Ejemplo

Escribamos al $mcd(32, 17)$ como combinación lineal entre ellos:

```
alfa( 1 , 0 )= 1, beta( 1 , 0 )= 0, mcd( 1 , 0 )= 1
1 = 1 * 1 + 0 * 0
alfa( 2 , 1 )= 0, beta( 2 , 1 )= 1, mcd( 2 , 1 )= 1
1 = 0 * 2 + 1 * 1
alfa( 15 , 2 )= 1, beta( 15 , 2 )= -7, mcd( 15 , 2 )= 1
1 = 1 * 15 + -7 * 2
alfa( 17 , 15 )= -7, beta( 17 , 15 )= 8, mcd( 17 , 15 )= 1
1 = -7 * 17 + 8 * 15
alfa( 32 , 17 )= 8, beta( 32 , 17 )= -15, mcd( 32 , 17 )= 1
1 = 8 * 32 + -15 * 17
```

Otro Ejemplo

Escribamos al $mcd(360, 28) = 4$ como combinación lineal entre ellos:

```
alfa( 4 , 0 )= 1, beta( 4 , 0 )= 0, mcd( 4 , 0 )= 4
4 = 1 * 4 + 0 * 0
alfa( 24 , 4 )= 0, beta( 24 , 4 )= 1, mcd( 24 , 4 )= 4
4 = 0 * 24 + 1 * 4
alfa( 28 , 24 )= 1, beta( 28 , 24 )= -1, mcd( 28 , 24 )= 4
4 = 1 * 28 + -1 * 24
alfa( 360 , 28 )= -1, beta( 360 , 28 )= 13, mcd( 360 , 28 )= 4
4 = -1 * 360 + 13 * 28
```

¿y el programita ?

Para implementar esto en un programa de computadora (en Python 3) será conveniente en realidad implementar una funcion ecl que calcula la terna

$$(\alpha(a,b), \beta(a,b), \mathrm{mcd}(a,b))$$

recursivamente (¡todo al mismo tiempo!), dado que las definiciones de α y β no son independientes sino que están cruzadas.

Programa para el MCD como combinacion lineal

```
def ecl(a,b):
    if b>a:
     return ecl(b,a)
    if b==0:
         alfa_a_b=1
         beta_a_b=0
         mcd_a_b=a
    else:
        q,r = divmod(a,b)
        alfa_b_r, beta_b_r, mcd_b_r = ecl(b,r)
        alfa_a_b =  beta_b_r
        beta_a_b = alfa_b_r - beta_b_r * q
        mcd_a_b = mcd_b_r
    chequea_invariante (a,b,alfa_a_b,beta_a_b,mcd_a_b)
    return (alfa_a_b,beta_a_b,mcd_a_b)
```

Función que chequea el invariante del algoritmo

La condición

$$\alpha(a,b)\ a + \beta(a,b)\ b = mcd(a,b)$$

es ahora el invariante del algoritmo. La siguiente función (en el sentido informático del término), permite chequearla en cada paso. Es decir nos ayuda a comprobar la correctitud de nuestro algoritmo:

```
def chequea_invariante(a,b,alfa_a_b,beta_a_b,mcd_a_b):
  print("alfa(",a,",",b,")=",alfa_a_b,end=', ')
  print("beta(",a,",",b,")=",beta_a_b,end=', ')
  print("mcd(",a,",",b,")=",mcd_a_b)
  print(mcd_a_b,"=",alfa_a_b,"*",a,"+",beta_a_b,"*",b)
```

Parte III

Bonus track: ¿cuánto tarda el algoritmo de Euclides?

Los números de Fibonacci

Recordamos que los números de Fibonacci se definen recursivamente por

$$F_0 = 0, F_1 = 1, F_n = F_{n-1} + F_{n-2} \quad (n \geq 2)$$

Es fácil escribir un programa para generar los números de Fibonacci

```
def fibo(n):
  if n==0:
    return 0
  elif n==1:
    return 1
  else:
    return fibo(n-1)+fibo(n-2)

for k in range(0,16):
  print ("fibo(",k,")=",fibo(k))
```

Tabla de los números de Fibonacci

```
fibo( 0 )= 0
fibo( 1 )= 1
fibo( 2 )= 1
fibo( 3 )= 2
fibo( 4 )= 3
fibo( 5 )= 5
fibo( 6 )= 8
fibo( 7 )= 13
fibo( 8 )= 21
fibo( 9 )= 34
fibo( 10 )= 55
fibo( 11 )= 89
fibo( 12 )= 144
fibo( 13 )= 233
fibo( 14 )= 377
fibo( 15 )= 610
```

¿Qué pasa si aplicamos el algoritmo de Euclides a dos números de Fibonacci consecutivos?

Calculemos $\text{mcd}(F_k, F_{k+1})$ para distintos valores de k usando nuestros programitas:
Por ejemplo si $k = 11$:

```
a=fibo( 12 )= 144
b=fibo( 11 )= 89
144 = 1 * 89 + 55
89 = 1 * 55 + 34
55 = 1 * 34 + 21
34 = 1 * 21 + 13
21 = 1 * 13 + 8
13 = 1 * 8 + 5
8 = 1 * 5 + 3
5 = 1 * 3 + 2
3 = 1 * 2 + 1
2 = 2 * 1 + 0
```

Generalizando....

Notamos que:

- Se requieren exactamente $k-1$ pasos para calcular $\text{mcd}(F_k, F_{k+1})$ usando el algoritmo de Euclides.
- Todos los cocientes son 1 (salvo el último) y los restos son los números de Fibonacci.
- Se puede ver que este es el peor caso del algoritmo de Euclides, o sea: que si $b \leq a \leq F_{k+1}$ el algoritmo va a tardar menos. (Porque los cocientes son q_j por lo menos 1, y si son más grandes, el algoritmo tarda menos)

La complejidad del algoritmo de Euclides

- Pero

$$F_{k+1} \simeq \frac{1}{\sqrt{5}} \Phi^{k+1}$$

si k es grande, donde

$$\Phi = \frac{1+\sqrt{5}}{2} \approx 1,61803$$

es el número de oro. Se deduce que si a es grande y $b \leq a$, el número de divisiones que requiere el algoritmo de Euclides para calcular $\text{mcd}(a, b)$ es aproximadamente (en el peor caso)

$$\frac{\log\ a}{\log\ \Phi}$$

(¡Esto es muy bueno!, tiene complejidad lineal como función del número de dígitos de a)