

Máximo divisor comum

O **máximo divisor comum** de dois números inteiros m e n , onde pelo menos um é não nulo, é o maior divisor comum de m e n .

O máximo divisor comum de m e n é denotado por $\text{mdc}(m, n)$.

Exemplo:

máximo divisor comum de 30 e 24 é 6

máximo divisor comum de 514229 e 317811 é 1

máximo divisor comum de 3267 e 2893 é 11

◀ ▶ ⏪ ⏩ 🔍 ↺

Café com leite

Recebe números inteiros não-negativos m e n e devolve $\text{mdc}(m, n)$. Supõe $m, n > 0$.

```
#define min(m,n) ((m) < (n) ? (m) : (n))
int mdc(int m, int n)
{
    int d = min(m,n);
    while (/*1*/ m % d != 0 || n % d != 0)
        /*2*/
        d--;
    /*3*/
    return d;
}
```

◀ ▶ ⏪ ⏩ 🔍 ↺

Invariantes e correção

Eis **relações invariantes** para a função mdc .

Em **/*1*/** vale que

(i0) $1 \leq d \leq \min(m, n)$, e

(i1) $m \% t \neq 0$ ou $n \% t \neq 0$ para todo $t > d$,

e em **/*2*/** vale que

(i2) $m \% d \neq 0$ ou $n \% d \neq 0$.

◀ ▶ ⏪ ⏩ 🔍 ↺

Máximo divisor comum

Problema: Dados dois números inteiros não-negativos m e n , determinar $\text{mdc}(m, n)$.

Exemplo:

máximo divisor comum de 30 e 24 é 6

máximo divisor comum de 514229 e 317811 é 1

máximo divisor comum de 3267 e 2893 é 11

◀ ▶ ⏪ ⏩ 🔍 ↺

Invariantes e correção

Passamos agora a verificar a **correção do algoritmo**.

Correção da função = a função funciona = a função faz o que promete.

A correção de **algoritmos iterativos** é comumente baseada na demonstração da validade de **invariantes**.

Estes **invariantes** são afirmações ou **relações** envolvendo os objetos mantidos pelo algoritmo.

◀ ▶ ⏪ ⏩ 🔍 ↺

Invariantes e correção

É evidente que em **/*3*/**, antes da função retornar d , vale que

$$m \% d = 0 \text{ e } n \% d = 0.$$

Como (i1) vale em **/*1*/**, então (i1) também vale em **/*3*/**. Assim, nenhum número inteiro maior que o valor d retornado divide m e n . Portanto, o valor retornado é de fato o $\text{mdc}(m, n)$.

Invariantes são assim mesmo. A validade de alguns torna a correção do algoritmo (muitas vezes) evidente. Os invariantes secundários servem para confirmar a validade dos principais.

◀ ▶ ⏪ ⏩ 🔍 ↺

Correção

A correção da recorrência proposta por Euclides é baseada no seguinte fato.

Se m, n e d são números inteiros, $m \geq 0$,
 $n, d > 0$, então

d divide m e $n \iff d$ divide n e $m\%n$.

Euclides recursivo

```
int euclidesR(int m, int n)
{
    if (n == 0) return m;
    return euclidesR(n, m % n);
}
```

Euclides iterativo

```
int euclidesI(int m, int n) {
    int r;
    do
    {
        r = m % n;
        m = n;
        n = r;
    }
    while (r != 0);
    return m;
}
```

euclidesR(317811,514229)

```
euclidesR(317811,514229)
euclidesR(514229,317811)
euclidesR(317811,196418)
euclidesR(196418,121393)
euclidesR(121393,75025)
euclidesR(75025,46368)
euclidesR(46368,28657)
euclidesR(28657,17711)
euclidesR(17711,10946)
euclidesR(10946,6765)
euclidesR(6765,4181)
euclidesR(4181,2584)
euclidesR(2584,1597)
euclidesR(1597,987)
euclidesR(987,610)
euclidesR(610,377)
euclidesR(377,233)
euclidesR(233,144)
euclidesR(144,89)
euclidesR(89,55)
euclidesR(55,34)
euclidesR(34,21)
euclidesR(21,13)
euclidesR(13,8)
euclidesR(8,5)
euclidesR(5,3)
euclidesR(3,2)
euclidesR(2,1)
euclidesR(1,0)

mdc(317811,514229) = 1.
```

Qual é mais eficiente?

```
meu_prompt>time ./mdc 317811 514229
mdc(317811,514229)=1
real 0m0.004s
user 0m0.004s
sys 0m0.000s
```

```
meu_prompt>time ./euclidesR 317811 514229
mdc(317811,514229)=1
real 0m0.002s
user 0m0.000s
sys 0m0.000s
```

Qual é mais eficiente?

```
meu_prompt>time ./mdc 2147483647 2147483646
mdc(2147483647,2147483646)=1
real 0m11.366s
user 0m11.361s
sys 0m0.000s
```

```
meu_prompt>time ./euclidesR 2147483647 2147483646
mdc(2147483647,2147483646)=1
real 0m0.002s
user 0m0.000s
sys 0m0.000s
```

Consumo de tempo

O consumo de tempo da função `euclidesR` é proporcional ao número de chamadas recursivas.

Suponha que `euclidesR` faz k chamadas recursivas e que no início da 1a. chamada ao algoritmo tem-se que $0 < n \leq m$.

Sejam

$$(m, n) = (m_0, n_0), (m_1, n_1), \dots, (m_k, n_k) = (\text{mdc}(m, n), 0),$$

os valores dos parâmetros no início de cada uma das chamadas da função.

◀ ▶ ↻ 🔍

Número de chamadas recursivas

Estimaremos o valor de k em função de $n = \min(m, n)$.

Note que $m_{i+1} = n_i$ e $n_{i+1} = m_i \% n_i$ para $i=1, 2, \dots, k$.

Note ainda que para inteiros a e b , $0 < b \leq a$ vale que

$$a \% b < \frac{a}{2} \quad (\text{verifique!}).$$

◀ ▶ ↻ 🔍

Número de chamadas recursivas

Seja t o número inteiro tal que

$$2^t \leq n < 2^{t+1}$$

Da primeira desigualdade temos que

$$t \leq \lg n,$$

onde $\lg n$ denota o logaritmo de n na base 2.

Da desigualdade estrita, concluímos que

$$k \leq 2(t + 1) - 1 = 2t + 1$$

Logo, o número k de chamadas recursivas é não superior a

$$2t + 1 \leq 2 \lg n + 1.$$

◀ ▶ ↻ 🔍

Número de chamadas recursivas

Por exemplo, para $m = 514229$ e $n = 317811$ tem-se

$$(m_0, n_0) = (514229, 317811),$$

$$(m_1, n_1) = (317811, 196418),$$

$$(m_2, n_2) = (196418, 121393),$$

$$\dots = \dots$$

$$(m_{27}, n_{27}) = (1, 0).$$

◀ ▶ ↻ 🔍

Número de chamadas recursivas

Desta forma tem-se que

$$n_2 = m_1 \% n_1 = n_0 \% n_1 < n_0/2 = n/2 = n/2^1$$

$$n_4 = m_3 \% n_3 = n_2 \% n_3 < n_2/2 < n/4 = n/2^2$$

$$n_6 = m_5 \% n_5 = n_4 \% n_5 < n_4/2 < n/8 = n/2^3$$

$$n_8 = m_7 \% n_7 = n_6 \% n_7 < n_6/2 < n/16 = n/2^4$$

$$n_{10} = m_9 \% n_9 = n_8 \% n_9 < n_8/2 < n/32 = n/2^5$$

...

...

...

Percebe-se que depois de cada 2 chamadas recursivas o valor do segundo parâmetro é reduzido a menos da sua metade.

◀ ▶ ↻ 🔍

Número de chamadas recursivas

Para o exemplo acima, onde $m=514229$ e $n=317811$, temos que

$$2 \lg n + 1 = 2 \lg(317811) + 1 < 2 \times 18,3 + 1 < 37,56$$

e o número de chamadas recursivas feitas por `euclidesR(514229,317811)` foram 27.

◀ ▶ ↻ 🔍

