Introdução à Programação

1.Ano LCC-MIERSI
DCC - FCUP

Nelma Moreira

Aula 4

Análise de propriedades duma sequência de inteiros lidos (ou gerados) um a um

Análise de propriedades duma sequência de inteiros lidos (ou gerados) um a um

* contar o número de elementos com uma dada propriedade. Ex: serem pares, serem múltiplos (divisores) do primeiro elemento, serem maiores que o primeiro, etc.

Análise de propriedades duma sequência de inteiros lidos (ou gerados) um a um

* contar o número de elementos com uma dada propriedade. Ex: serem pares, serem múltiplos (divisores) do primeiro elemento, serem maiores que o primeiro, etc.

* determinar o maior e o menor com uma dada propriedade

Análise de propriedades duma sequência de inteiros lidos (ou gerados) um a um

* contar o número de elementos com uma dada propriedade. Ex: serem pares, serem múltiplos (divisores) do primeiro elemento, serem maiores que o primeiro, etc.
* determinar o maior e o menor com uma dada propriedade
* somar os números com uma dada propriedade. Ex: serem positivos, pares, etc

Algoritmo Genérico

Algoritmo Genérico

* Ler os números um a um;

Algoritmo Genérico

* Ler os números um a um;
* para cada número lido verificar a propriedade e se a satisfazer efectuar a tarefa pretendida.

Mais Propriedades

Mais Propriedades

* o tamanho da maior subsequência de inteiros consecutivos com uma dada propriedade. Ex:serem pares, serem iguais, etc.

Mais Propriedades

* o tamanho da maior subsequência de inteiros consecutivos com uma dada propriedade. Ex:serem pares, serem iguais, etc.
* o tamanho da maior subsequência não crescente

Mais Propriedades

* o tamanho da maior subsequência de inteiros consecutivos com uma dada propriedade. Ex:serem pares, serem iguais, etc.
* o tamanho da maior subsequência não crescente
* contar o número de subsequências (com mais que um elemento) com uma dada propriedade. Ex: serem crescentes, só terem pares, etc.

Mais Propriedades

* o tamanho da maior subsequência de inteiros consecutivos com uma dada propriedade. Ex:serem pares, serem iguais, etc.
* o tamanho da maior subsequência não crescente
* contar o número de subsequências (com mais que um elemento) com uma dada propriedade. Ex: serem crescentes, só terem pares, etc.
* verificar se a sequência é crescente

Algoritmo Genérico

Algoritmo Genérico

* Ler os números um a um, guardando sempre o anterior;

Algoritmo Genérico

* Ler os números um a um, guardando sempre o anterior;
* para cada número lido comparar com o anterior;

Algoritmo Genérico

* Ler os números um a um, guardando sempre o anterior;
* para cada número lido comparar com o anterior;
* verificar se a propriedade se mantêm ou não;

Algoritmo Genérico

* Ler os números um a um, guardando sempre o anterior;
* para cada número lido comparar com o anterior;
* verificar se a propriedade se mantêm ou não;
* senão reiniciar uma nova contagem e testar a propriedade.

E mais alguns...

E mais alguns...

* contar as ocorrências de um mesmo número, em posições particulares. Ex: ímpares ou pares consecutivas,isto é, se o 1 é igual ao 3, o 2 é igual ao 4, o 3 é igual ao 5, ...

E mais alguns...

* contar as ocorrências de um mesmo número, em posições particulares. Ex: ímpares ou pares consecutivas,isto é, se o 1 é igual ao 3, o 2 é igual ao 4, o 3 é igual ao 5, ...

* se são ou gerar termos de uma sucessão num dado intervalo. Ex: sucessão dos pares, dos múltiplos de um dado número, da forma 2n, uma sucessão aritmética, etc,

E mais alguns...

* contar as ocorrências de um mesmo número, em posições particulares. Ex: ímpares ou pares consecutivas,isto é, se o 1 é igual ao 3, o 2 é igual ao 4, o 3 é igual ao 5, ...

* se são ou gerar termos de uma sucessão num dado intervalo. Ex: sucessão dos pares, dos múltiplos de um dado número, da forma 2n, uma sucessão aritmética, etc,

$$u_n = f(u_{n-1}, \ldots, u_{n_k}) \text{ e } u_1, \ldots, u_k \text{ valores dados}$$

Algoritmo genérico para a geração

Algoritmo genérico para a geração

* São necessárias k+1 variáveis que guardem sempre os últimos k termos e o próximo.

Algoritmo genérico para a geração

* São necessárias k+1 variáveis que guardem sempre os últimos k termos e o próximo.
* Obtém-se o valor seguinte usando a expressão e escreve-se.

Algoritmo genérico para a geração

* São necessárias k+1 variáveis que guardem sempre os últimos k termos e o próximo.
* Obtém-se o valor seguinte usando a expressão e escreve-se.
* Actualizam-se os últimos k termos.

Como termina a sequência?

Como termina a sequência?

* sabendo o número de elementos no início e usando um contador

Como termina a sequência?

* sabendo o número de elementos no início e usando um contador
* supondo que é terminada por um valor especial (sentinela)

Contar os múltiplos do primeiro

Contar os múltiplos do primeiro

Supor que a sequência termina por zero.

Contar os múltiplos do primeiro

Supor que a sequência termina por zero.

```
main(){
   int i,m,c = 0;
   scanf("%d",&i);
   m = i;
   while(i != 0) {
if(i%m == 0) c++;
    scanf("%d",&i);
   }
 printf("%d",c);
}
```

Somar os maiores que o primeiro

Somar os maiores que o primeiro

Supor que a sequência termina por zero.

Somar os maiores que o primeiro

Supor que a sequência termina por zero.

```
main(){
   int i,m,s = 0;
   scanf("%d",&i);
   m = i;
   while(i != 0) {
      if(i>m) s += i;
      scanf("%d",&i);
   }
 printf("%d",s);
}
```

O tamanho da maior subsequência não crescente

O tamanho da maior subsequência não crescente

```
int main(){
  int i,ant,max=1,c=1;
  scanf("%d",&ant);
  if (!ant) return 0;
  scanf("%d",&i);
  while(i != 0){
    if(i <= ant) c++;
    else {
      if(max < c) {max = c;}
      c = 1;
    }
    ant = i;
    scanf("%d",&i);
  }
  if(max < c)max = c;
  printf("max=%d\n",max);
  return 1;}
```

Sucessão de Fibonacci

Sucessão de Fibonacci

Considerar uma população de coelhos que se reproduz segundo as seguintes regras:

Sucessão de Fibonacci

Considerar uma população de coelhos que se reproduz segundo as seguintes regras:

* Cada par de coelhos produz um novo par por mês

Sucessão de Fibonacci

Considerar uma população de coelhos que se reproduz segundo as seguintes regras:

* Cada par de coelhos produz um novo par por mês
* Os coelhos são férteis a partir do segundo mês

Sucessão de Fibonacci

Considerar uma população de coelhos que se reproduz segundo as seguintes regras:

* Cada par de coelhos produz um novo par por mês
* Os coelhos são férteis a partir do segundo mês
* Os coelhos não morrem

Sucessão de Fibonacci

Considerar uma população de coelhos que se reproduz segundo as seguintes regras:

* Cada par de coelhos produz um novo par por mês
* Os coelhos são férteis a partir do segundo mês
* Os coelhos não morrem

Supondo que nasce um par de coelhos em Janeiro, quantos pares de coelhos existem no fim do ano?

Algoritmo

Algoritmo

Determinar o número de pares em cada mês:

Algoritmo

Determinar o número de pares em cada mês:

0	1	2	3	4	5	6	7	8	9	10	11	12	11
0	1	1	2	3	5	8	13	21	34	55	89	144	

Algoritmo

Determinar o número de pares em cada mês:

0	1	2	3	4	5	6	7	8	9	10	11	12	11
0	1	1	2	3	5	8	13	21	34	55	89	144	

Generalizando, ao fim de n > 1 etapas temos:

Algoritmo

Determinar o número de pares em cada mês:

0	1	2	3	4	5	6	7	8	9	10	11	12	11
0	1	1	2	3	5	8	13	21	34	55	89	144	

Generalizando, ao fim de n > 1 etapas temos:

$$f_n = f_{n-1} + f_{n-2}, \text{ e } f_0 = 0, f_1 = 1$$

Pseudo-código

Pseudo-código

Variáveis

Pseudo-código

Variáveis

* z: termo corrente

Pseudo-código

Variáveis

* `z`: termo corrente
* `x,y`: termos anteriores

Pseudo-código

Variáveis

* z: termo corrente
* x, y: termos anteriores
* n : # termos da sucessão

Pseudo-código

Variáveis

* z: termo corrente
* x, y: termos anteriores
* n : # termos da sucessão

Pseudo-código

Variáveis

* z: termo corrente
* x, y: termos anteriores
* n : # termos da sucessão

Pseudo-código

Variáveis

* `z`: termo corrente

* `x,y`: termos anteriores

* `n` : # termos da sucessão

```
Ler n
```

Pseudo-código

Variáveis

* `z`: termo corrente

* `x,y`: termos anteriores

* `n` : # termos da sucessão

```
Ler n
x=0 e y=1
```

Pseudo-código

Variáveis

* z: termo corrente

* x,y: termos anteriores

* n : # termos da sucessão

```
Ler n
x=0 e y=1
Enquanto (n>0) faça {
```

Pseudo-código

Variáveis

* z: termo corrente
* x,y: termos anteriores
* n : # termos da sucessão

```
Ler n
x=0 e y=1
Enquanto (n>0) faça {
    z = x+y
```

Pseudo-código

Variáveis

* z: termo corrente
* x,y: termos anteriores
* n : # termos da sucessão

```
Ler n
x=0 e y=1
Enquanto (n>0) faça {
    z = x+y
    x = y
```

Pseudo-código

Variáveis

* z: termo corrente
* x,y: termos anteriores
* n : # termos da sucessão

```
Ler n
x=0 e y=1
Enquanto (n>0) faça {
    z = x+y
    x = y
    y = z
```

Pseudo-código

Variáveis

* z: termo corrente
* x, y: termos anteriores
* n : # termos da sucessão

```
Ler n
x=0 e y=1
Enquanto (n>0) faça {
    z = x+y
    x = y
    y = z
    n = n-1
```

Pseudo-código

Variáveis

* z: termo corrente
* x, y: termos anteriores
* n : # termos da sucessão

```
Ler n
x=0 e y=1
Enquanto (n>0) faça {
    z = x+y
    x = y
    y = z
    n = n-1
Escrever z
```

Pseudo-código

Variáveis

* z: termo corrente
* x, y: termos anteriores
* n : # termos da sucessão

```
Ler n
x=0 e y=1
Enquanto (n>0) faça {
    z = x+y
    x = y
    y = z
    n = n-1
Escrever z
```

O Programa em C

O Programa em C

```
main() {
   int x=0,y=1,z,n;
   scanf(“%d”, &n);
   printf("%d\n %d\n",x,y);
   while (n>0) {
      z=x+y;
      x=y;
      y=z;
      printf("%d \n",y);
      --n;
   }
```

Intervalo

5 minutos!

Primos

Primos

Determinar se um inteiro positivo é primo

Primos

Determinar se um inteiro positivo é primo

* Um número é primo se apenas é divisível por ele próprio e por 1.

Primos

Determinar se um inteiro positivo é primo

* Um número é primo se apenas é divisível por ele próprio e por 1.

* **Algoritmo:** percorrer os inteiros de 2 a n-1 e verificar se dividem n:

Primos

Determinar se um inteiro positivo é primo

* Um número é primo se apenas é divisível por ele próprio e por 1.

* **Algoritmo:** percorrer os inteiros de 2 a n-1 e verificar se dividem n:

    * se nenhum dividir n é primo!

Primos

Determinar se um inteiro positivo é primo

* Um número é primo se apenas é divisível por ele próprio e por 1.

* **Algoritmo:** percorrer os inteiros de 2 a n-1 e verificar se dividem n:

  * se nenhum dividir n é primo!

(1 não é nem composto nem primo!)

Primos

Determinar se um inteiro positivo é primo

* Um número é primo se apenas é divisível por ele próprio e por 1.

* **Algoritmo:** percorrer os inteiros de 2 a n-1 e verificar se dividem n:

  * se nenhum dividir n é primo!

(1 não é nem composto nem primo!)

Programa em C

```
#include <stdio.h>
main()  /* primalidade */
{
   int i = 2, n;
   scanf("%d",&n);
   while (n % i != 0 &&  i < n)  i++;
   if (i >= n) printf("%d e primo\n",n);
   else printf("%d nao e primo\n",n);
}
```

Melhorar a eficiência do algoritmo

Melhorar a eficiência do algoritmo

* verificar se é divisível por 2

Melhorar a eficiência do algoritmo

* verificar se é divisível por 2
* senão basta percorrer os ímpares...

Melhorar a eficiência do algoritmo

* verificar se é divisível por 2
* senão basta percorrer os ímpares...
* basta ir até n/2

Melhorar a eficiência do algoritmo

* verificar se é divisível por 2
* senão basta percorrer os ímpares...
* basta ir até n/2
* se tiver um divisor próprio tem de ser menor que `sqrt(n)` (porquê?)

Melhorar a eficiência do algoritmo

* verificar se é divisível por 2
* senão basta percorrer os ímpares...
* basta ir até n/2
* se tiver um divisor próprio tem de ser menor que `sqrt(n)` (porquê?)

Programa em C

```
#include <math.h>
#include <stdio.h>
int main()  {
   int i,n,md;
   printf("Introduzir inteiro: ");
   scanf("%d",&n);
   if (n < 1) {printf("não é positivo!"); return 0;}
   if(n == 2 || n == 3) { printf("primo\n");return 1;}
   if(n%2 == 0) printf("é par\n");
   else {
      md = sqrt(n);
      i = 3;
      while (n % i != 0 &&  i <= md) i += 2;
      if (i > md) printf("%d e primo\n",n);
   }
}
```

Eficiência do algoritmo

Eficiência do algoritmo

* Número de passos em função do tamanho dos dados

Eficiência do algoritmo

* Número de passos em função do tamanho dos dados
* Neste caso os dados são o valor `n` e o seu tamanho é dado pelo número de bits que ocupa (valor escrito na base dois): `log(n)`

Eficiência do algoritmo

* Número de passos em função do tamanho dos dados

* Neste caso os dados são o valor `n` e o seu tamanho é dado pelo número de bits que ocupa (valor escrito na base dois): `log(n)`

  porque `n = 2^(log(n))`

O algoritmo é exponêncial

O algoritmo é exponêncial

* O algoritmo tem de testar os inteiros de `2` a `n`, logo no pior caso (sem optimizações) são `n-2` passos (cerca de `n`)

O algoritmo é exponêncial

* O algoritmo tem de testar os inteiros de `2` a `n`, logo no pior caso (sem optimizações) são `n-2` passos (cerca de `n`)
* Se `n` tiver `x` bits, o tempo de execução é `t(x)=O(2^x)`

O algoritmo é exponêncial

* O algoritmo tem de testar os inteiros de `2` a `n`, logo no pior caso (sem optimizações) são `n-2` passos (cerca de `n`)
* Se `n` tiver `x` bits, o tempo de execução é `t(x)=O(2^x)`
* `O(f(x))` significa ordem de `f(x)`

Algoritmos exponênciais são maus!

Algoritmos exponênciais são maus!

```
n    2^n    n!
-------------------------------------------------------------------
1    2      1
11   2048   39916800
21   2097152  51090942171709440000
31   2147483648   8222838654177922817725562880000000
41   2199023255552  33452526613163807108170062053440751665152000000000
51   2251799813685248  155111875328738228022424301646930321106325972000
                       16986112000000000000
61   2305843009213693952   507580213877224798800856812176625227226004
                           528988036003099405939480985600000000000000
71   2361183241434822606848  850478588567862317521167644239926010288584
                             608120796235886430763388588680378079017697
                             2728000000000000000
81   241785163922925834941235 2  ....
91   247588007857076054979824844 8  135200152767840296255166568759495142
                                    214758686647690667779174173459715367
                                    077715599947656852839547504494277516
                                    8336768008192000000000000000000000
```

Factorização de um inteiro em factores primos

Factorização de um inteiro em factores primos

* Algoritmo I: gerar os primos inferiores a `n` e verificar se são divisores. Não é eficiente!

Factorização de um inteiro em factores primos

* Algoritmo I: gerar os primos inferiores a `n` e verificar se são divisores. Não é eficiente!

* Algoritmo II: se for divisível por `2`, fazer `n = n/2` e ir dividindo até deixar de ser e contando o número de divisões, repetir para `3, 5, 7, 9,...`

Factorização de um inteiro em factores primos

* Algoritmo I: gerar os primos inferiores a `n` e verificar se são divisores. Não é eficiente!

* Algoritmo II: se for divisível por `2`, fazer `n = n/2` e ir dividindo até deixar de ser e contando o número de divisões, repetir para `3, 5, 7, 9,...`

  Facto: o próximo ímpar que o divide é primo. Como no caso anterior, basta `n/2` ou melhor `sqrt(n)`.

Factorização de um inteiro em factores primos

* Algoritmo I: gerar os primos inferiores a `n` e verificar se são divisores. Não é eficiente!

* Algoritmo II: se for divisível por `2`, fazer `n = n/2` e ir dividindo até deixar de ser e contando o número de divisões, repetir para `3, 5, 7, 9,...`

  Facto: o próximo ímpar que o divide é primo. Como no caso anterior, basta `n/2` ou melhor `sqrt(n)`.

Programa em C

```
#include <math.h>
#include <stdio.h>
int main() {
  int i,n, md,p;
  printf("Introduzir
inteiro: ");
  scanf("%d",&n);
  printf("\n");
  if (n<1) {printf("nao e
positivo.\n");
  return 0;}
  if (n==2 || n==3) {
    printf("primo\n");
    return 1;}
  p=0;
  while (n%2==0) { p++;
    n=n/2;}
  if (p>0)
     printf("2^%d\n",p);
  if (n!=1) {
     md=sqrt(n); i=3;
     while (i<=md && n!=1)
     { p=0;
     while (n%i==0 && n!
     =1) { p++; n=n/i;}
     if (p>0) printf("%d ^
     %d\n",i,p);
     i+=2;
     }
     if (n!=1) printf("%d
\n",n);
}}
```

Problemas relacionados

* Primos de uma certa forma. P.e de Mersenne da forma $2^p - 1$ com p primo. Maior conhecido $p = 6972593$
* Dados dois inteiros determinar se um é divide o outro.
* Determinar todos os divisores de um inteiro.
* Números perfeitos: iguais à soma dos divisores próprios.
* Máximo divisor comum entre dois inteiros (Algoritmo de Euclides)

Algoritmo de Euclides

Algoritmo de Euclides

`mdc(m,n)`

Algoritmo de Euclides

`mdc(m,n)`

```
12
```

Algoritmo de Euclides

mdc(m,n)

12

Algoritmo de Euclides

mdc(m,n)

12

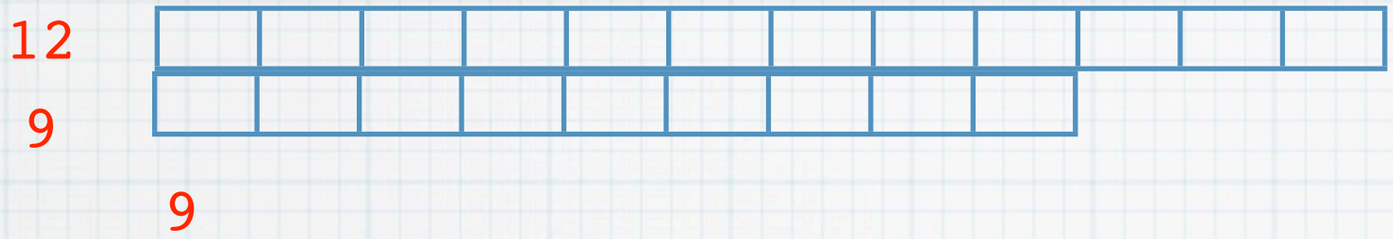
9

Algoritmo de Euclides

mdc(m,n)

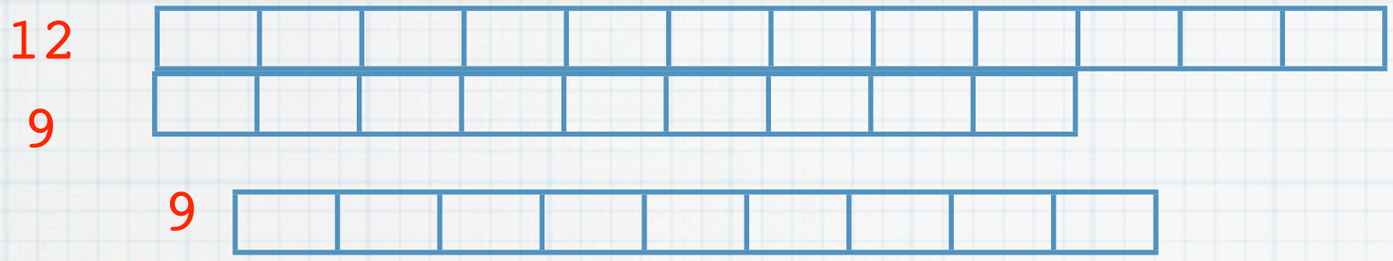
Algoritmo de Euclides

mdc(m,n)



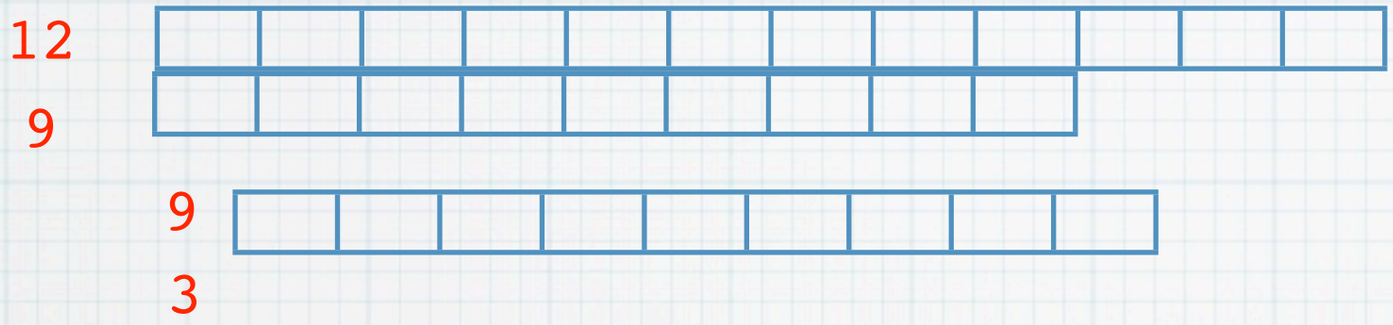
Algoritmo de Euclides

mdc(m,n)



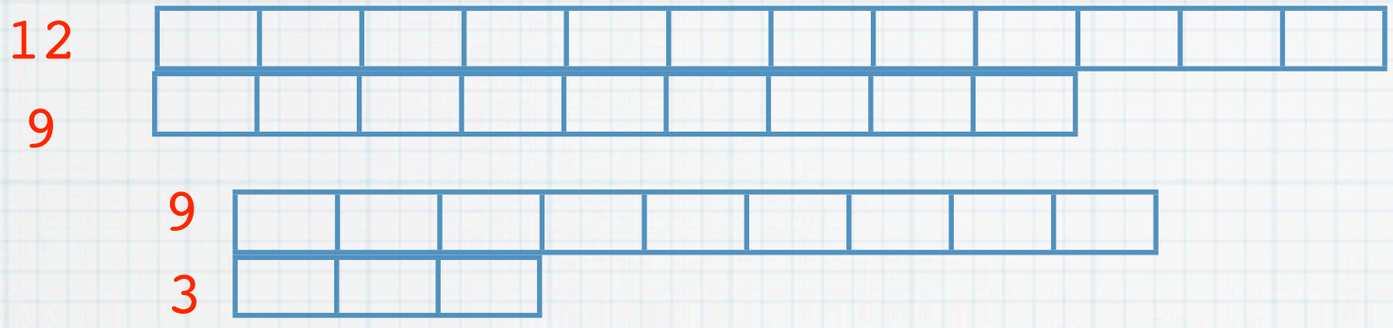
Algoritmo de Euclides

mdc(m,n)



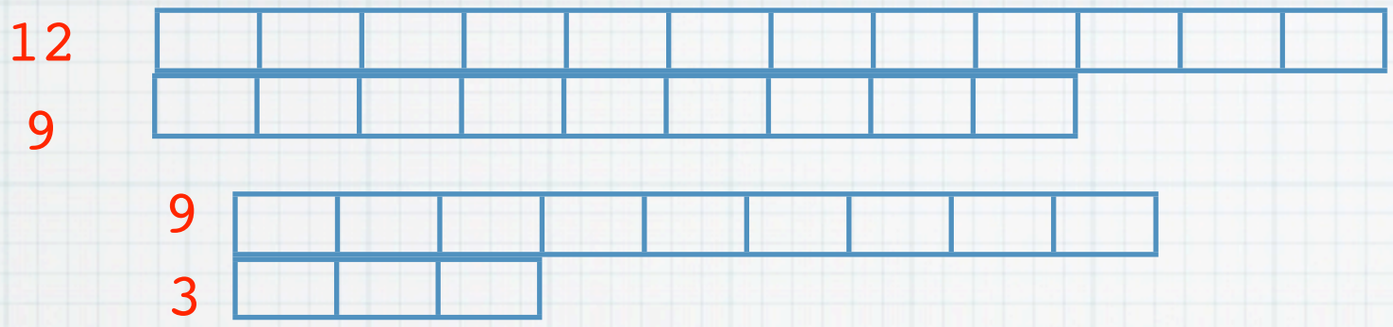
Algoritmo de Euclides

mdc(m,n)



Algoritmo de Euclides

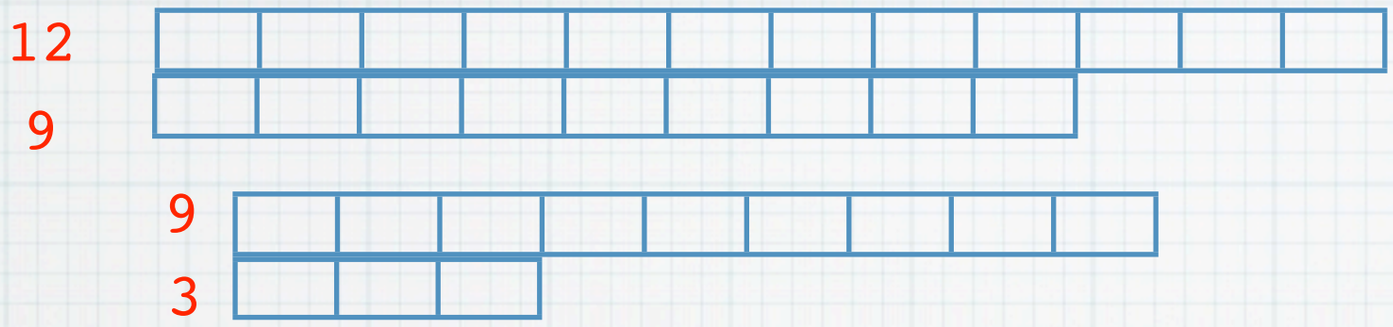
mdc(m,n)



mdc(12,9)=3

Algoritmo de Euclides

mdc(m,n)

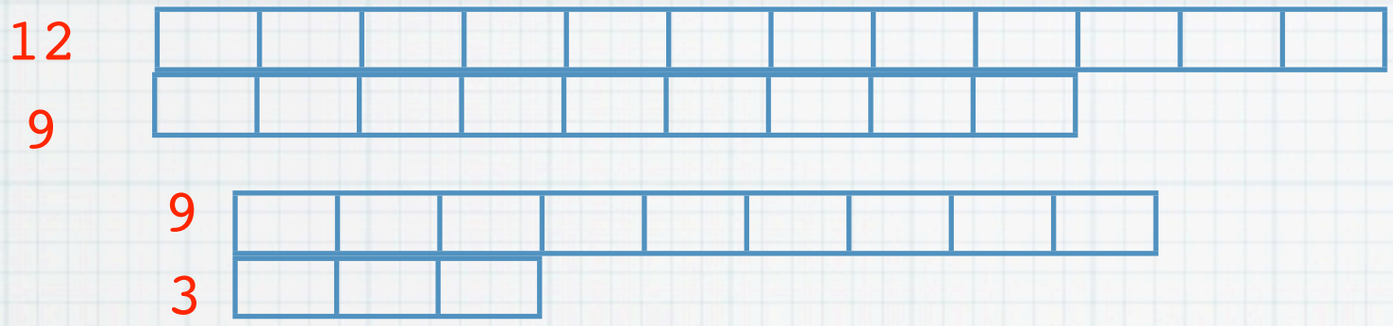


mdc(12,9)=3

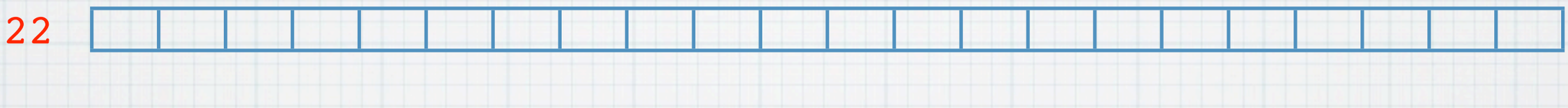
22

Algoritmo de Euclides

mdc(m,n)

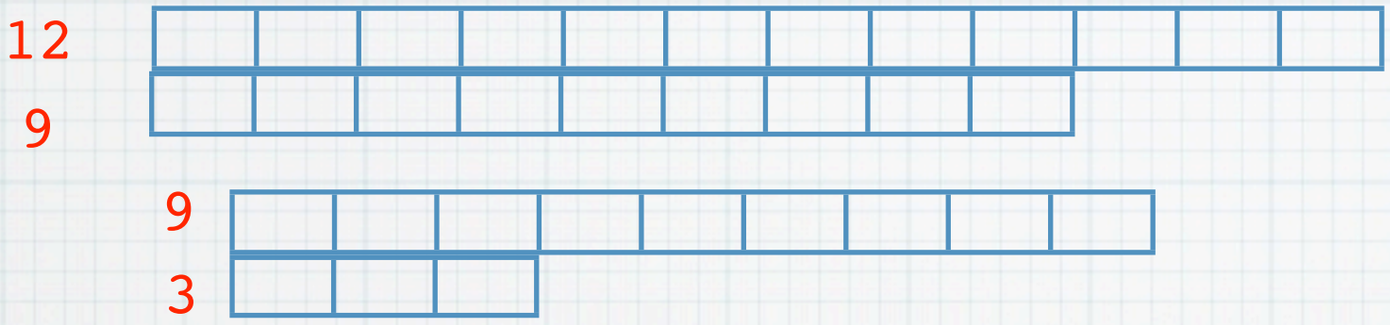


mdc(12,9)=3



Algoritmo de Euclides

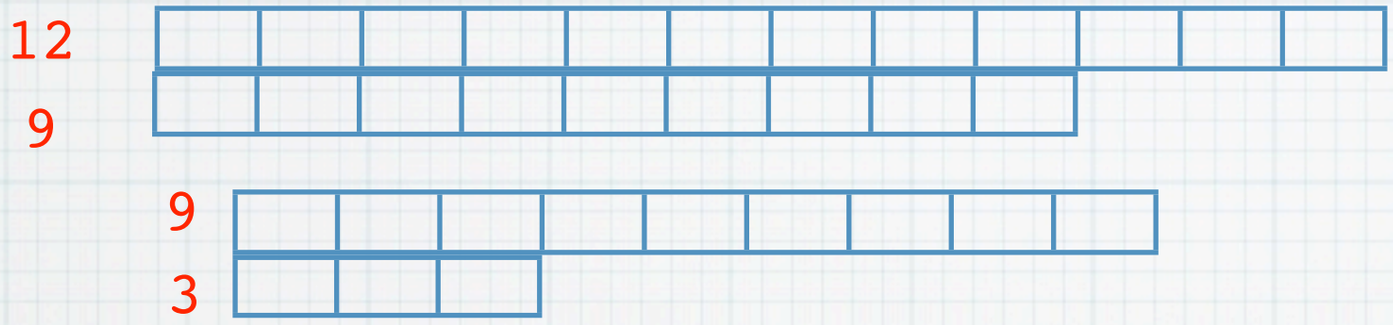
mdc(m,n)



mdc(12,9)=3

Algoritmo de Euclides

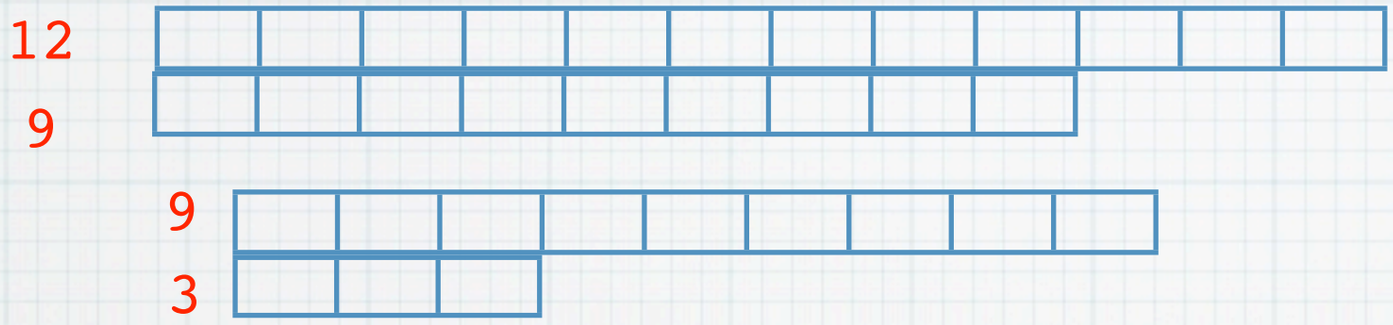
mdc(m,n)



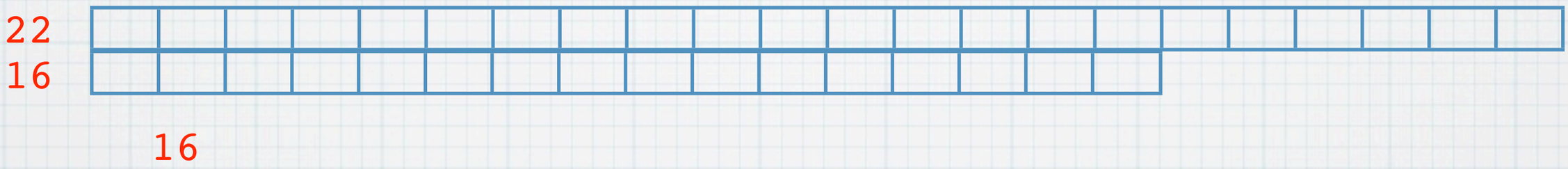
mdc(12,9)=3

Algoritmo de Euclides

mdc(m,n)

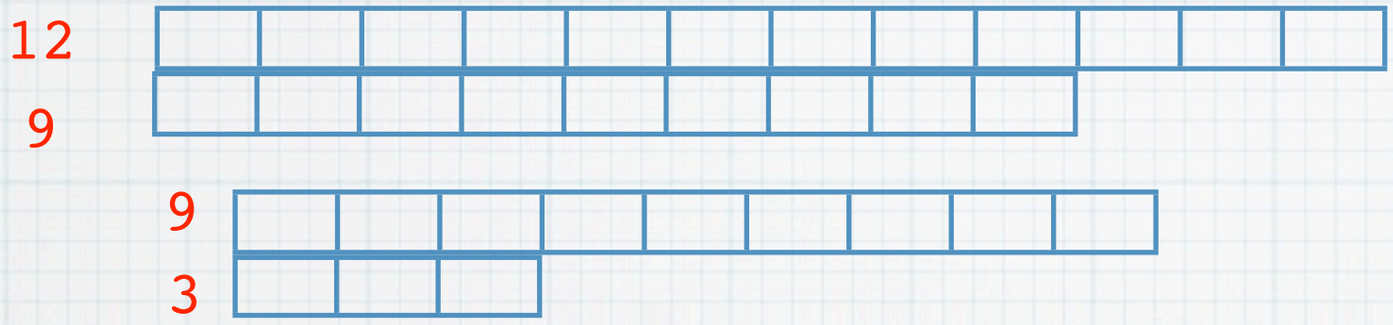


mdc(12,9)=3

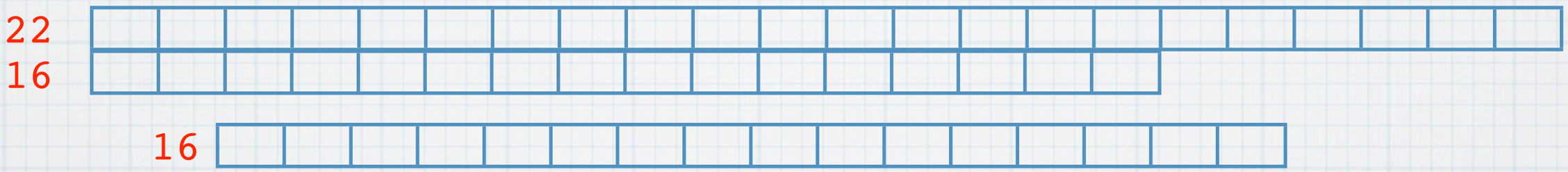


Algoritmo de Euclides

mdc(m,n)

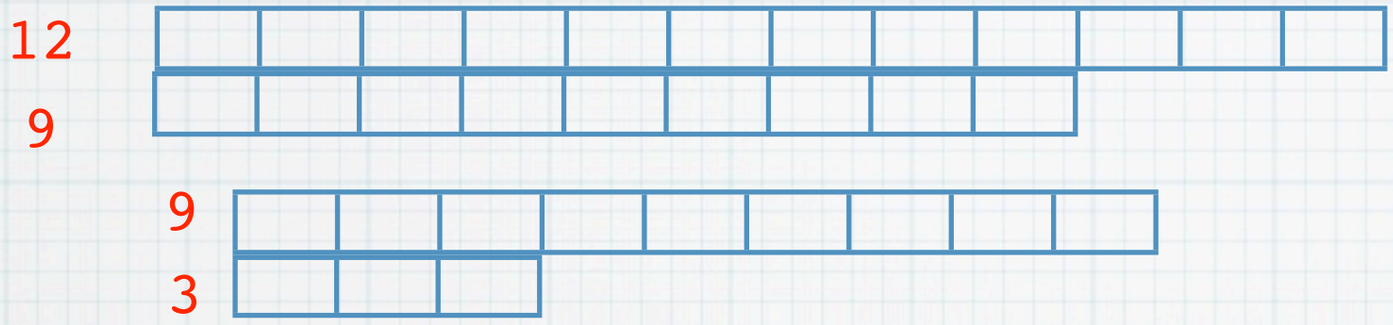


mdc(12,9)=3

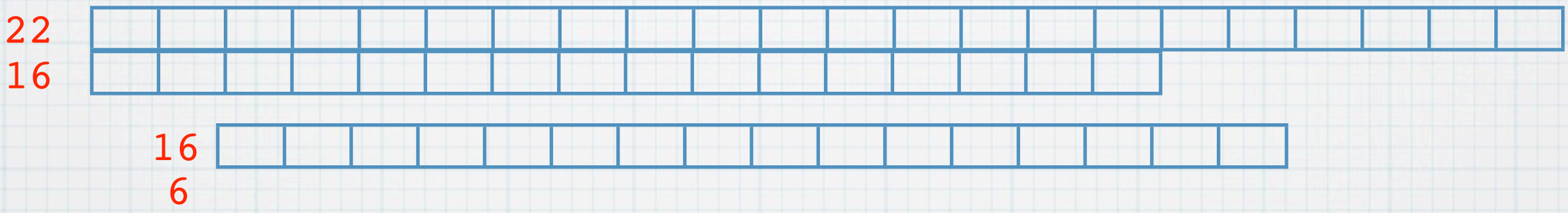


Algoritmo de Euclides

mdc(m,n)

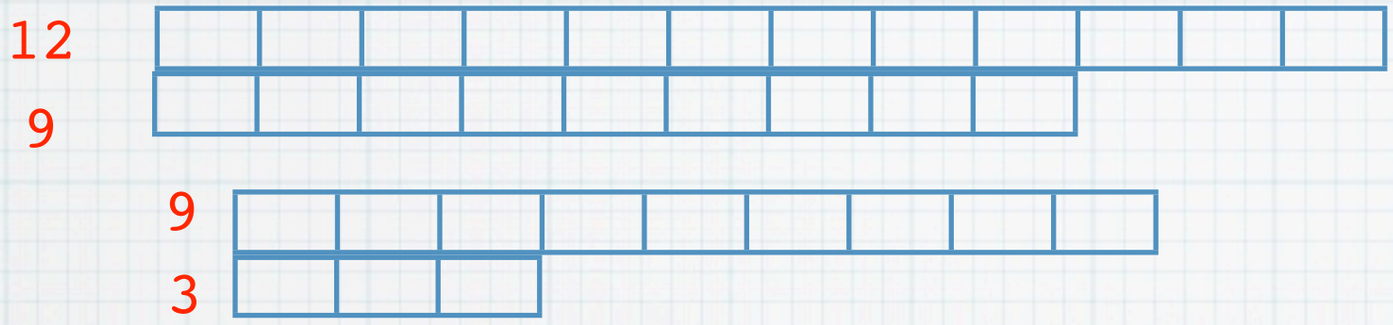


mdc(12,9)=3

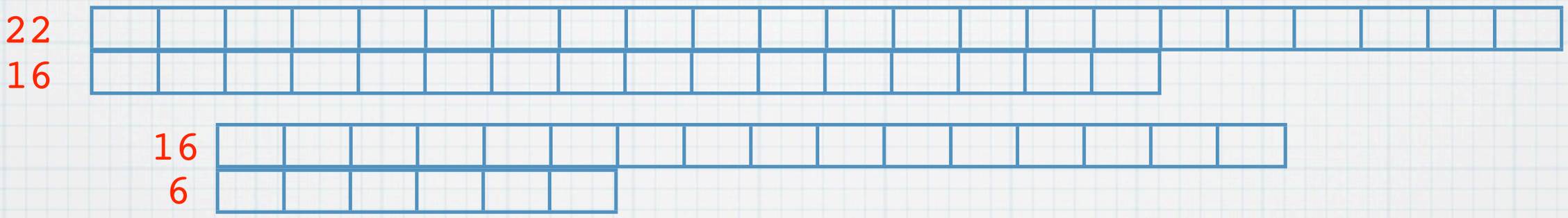


Algoritmo de Euclides

mdc(m,n)

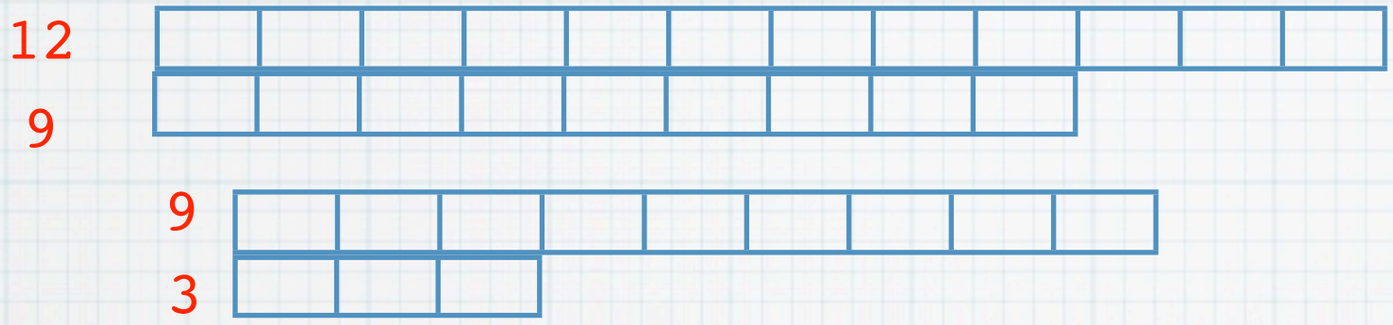


mdc(12,9)=3

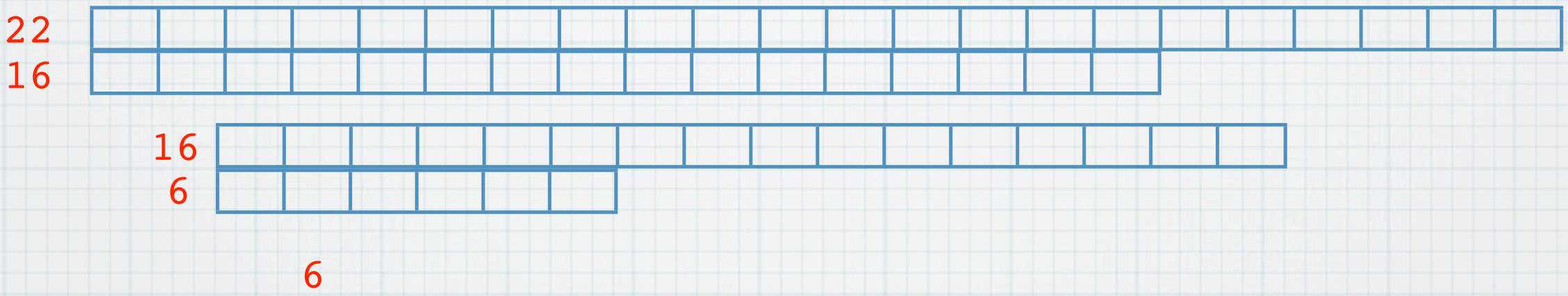


Algoritmo de Euclides

mdc(m,n)

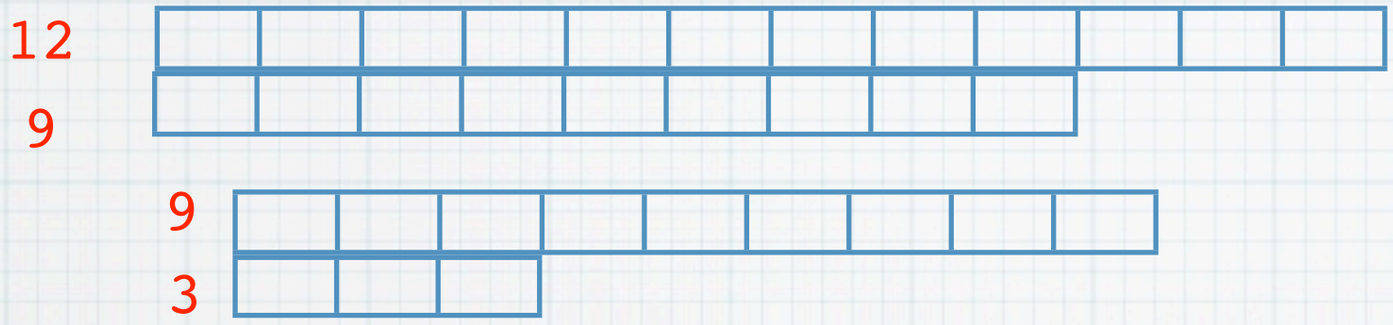


mdc(12,9)=3

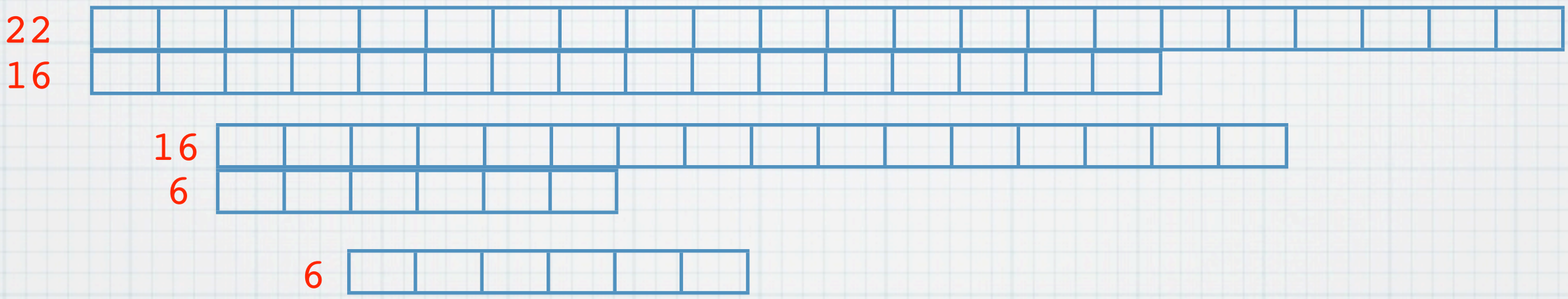


Algoritmo de Euclides

mdc(m,n)

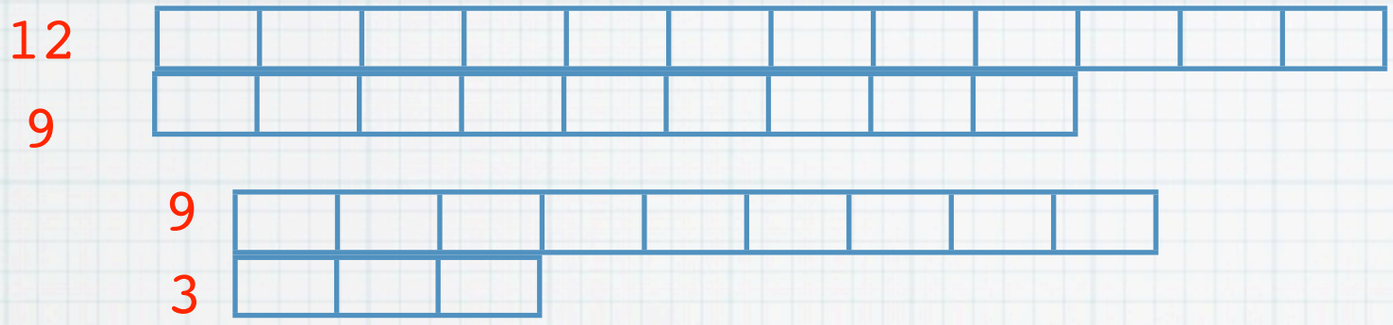


mdc(12,9)=3

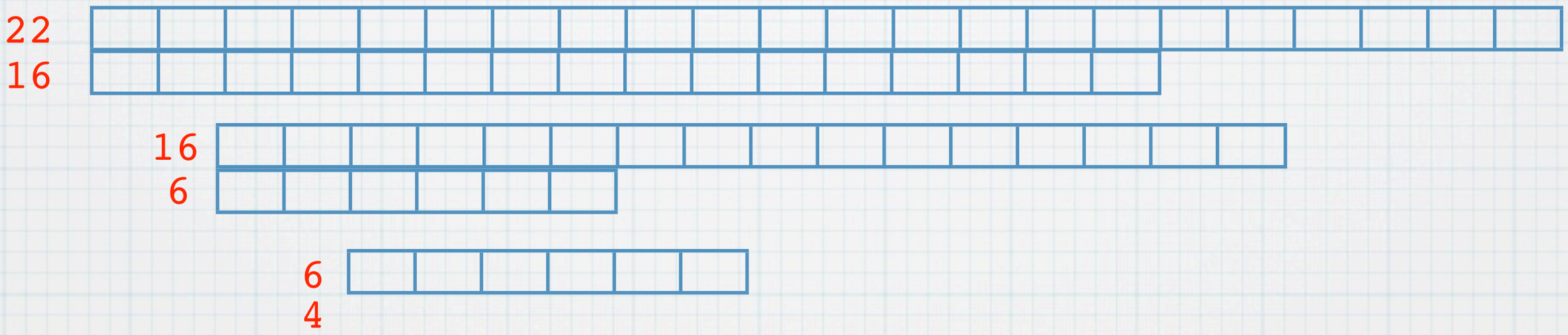


Algoritmo de Euclides

mdc(m,n)

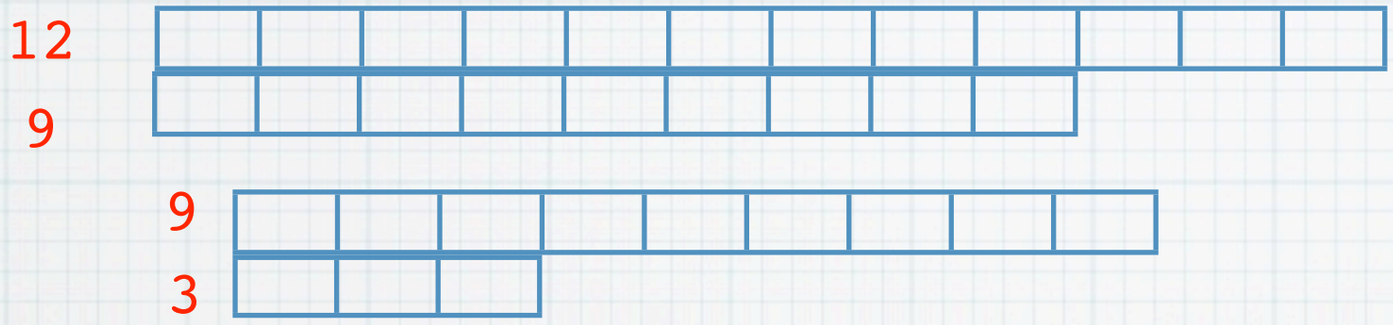


mdc(12,9)=3

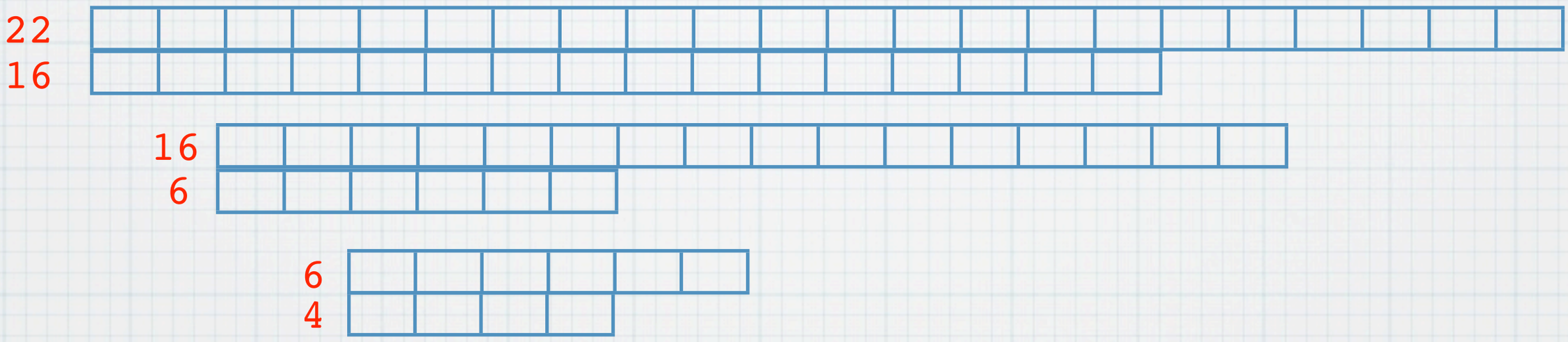


Algoritmo de Euclides

mdc(m,n)

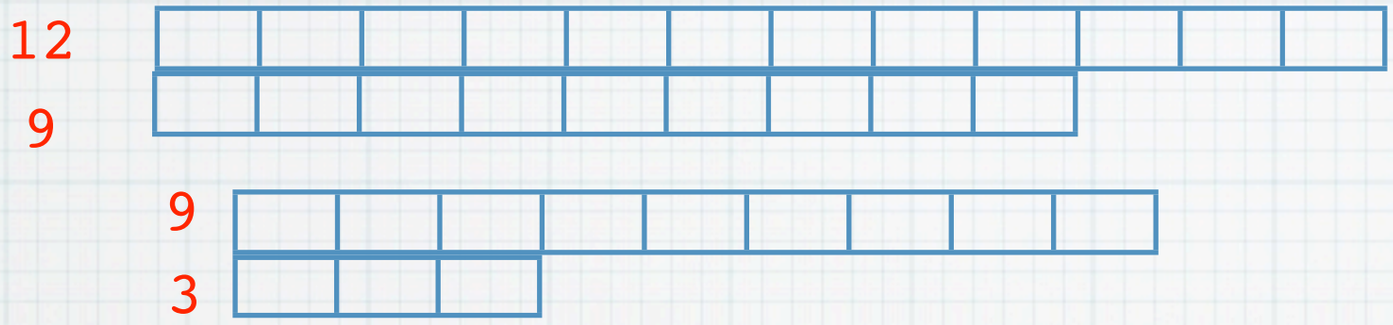


mdc(12,9)=3

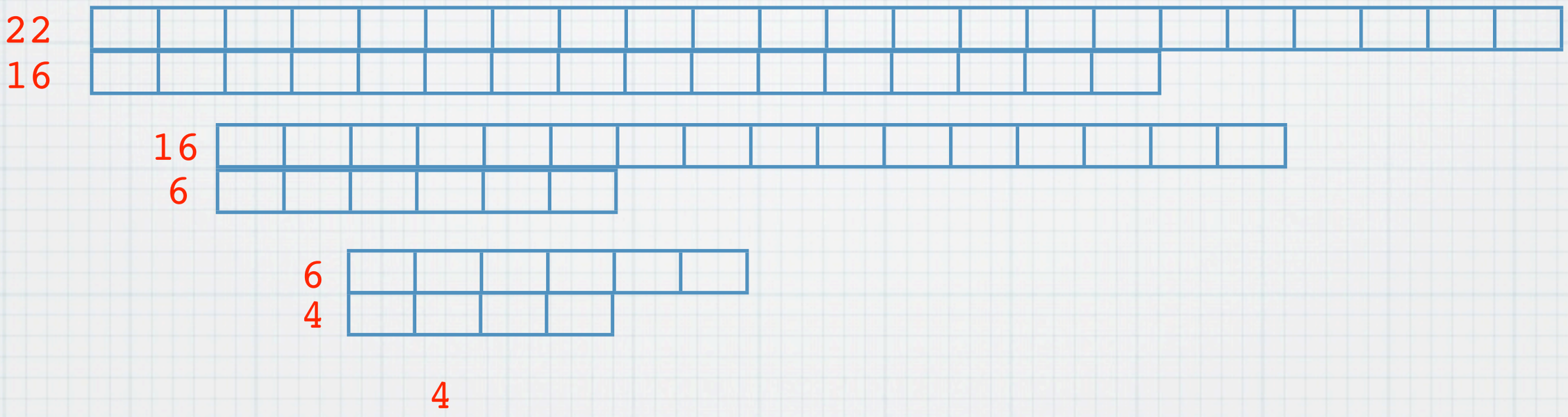


Algoritmo de Euclides

mdc(m,n)

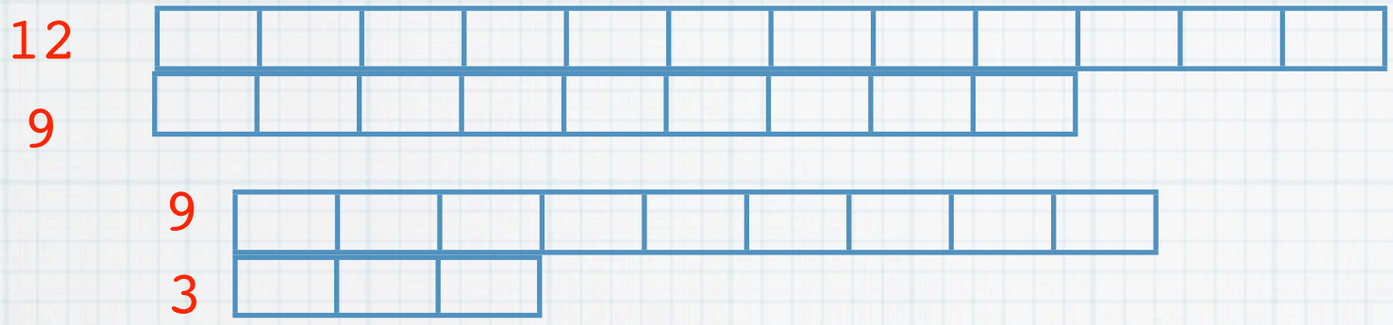


mdc(12,9)=3

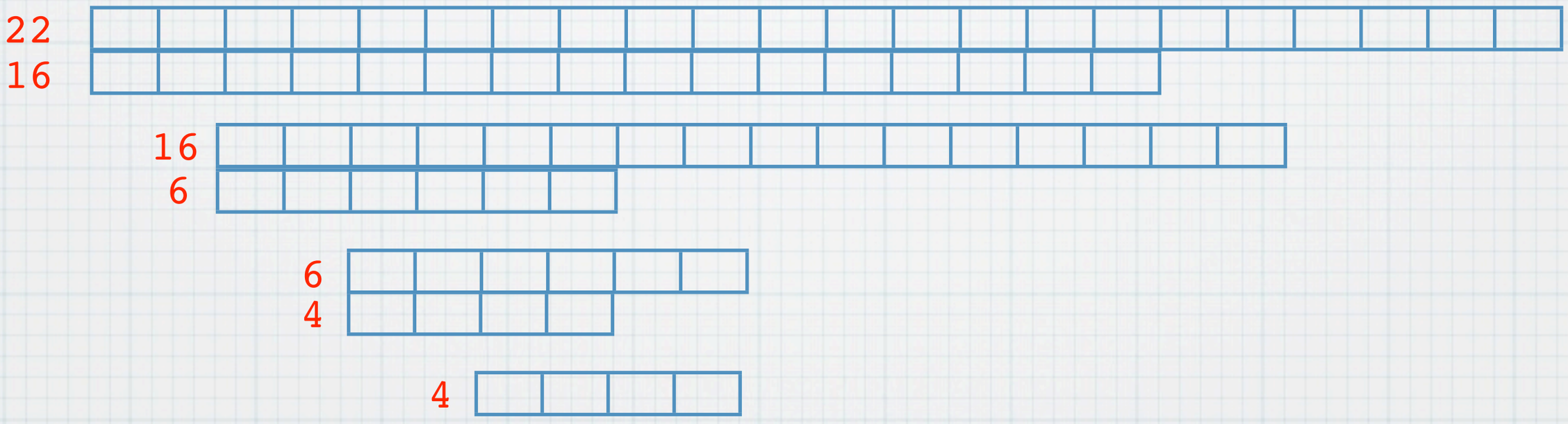


Algoritmo de Euclides

mdc(m,n)

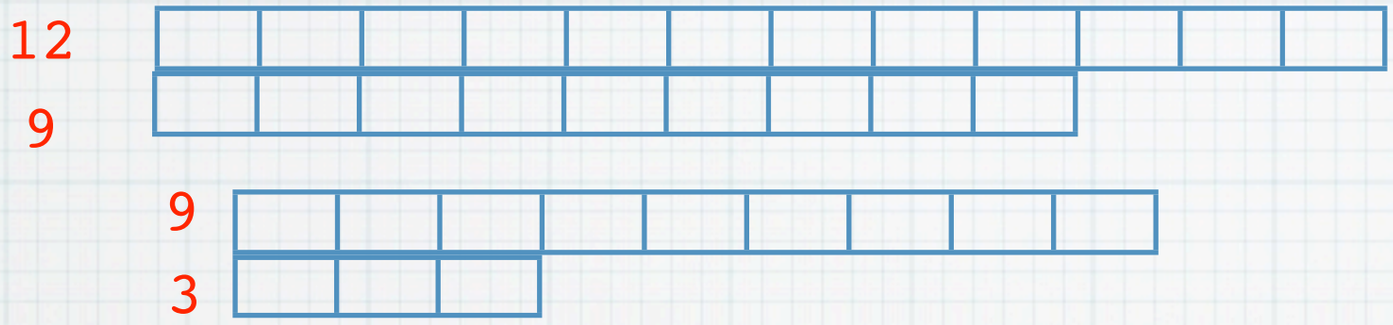


mdc(12,9)=3

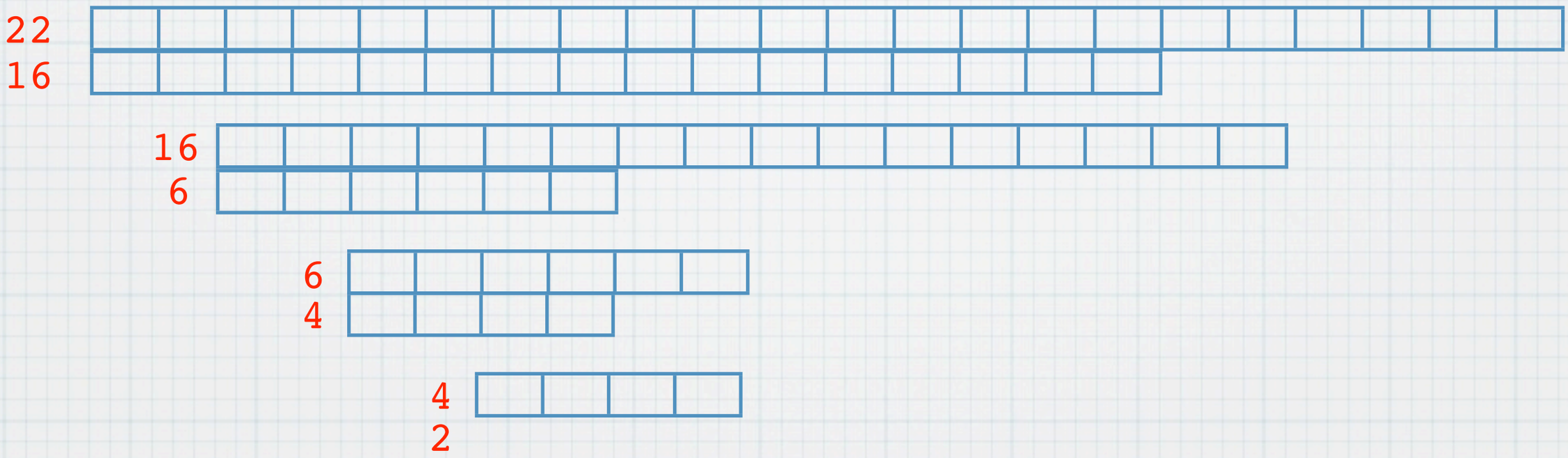


Algoritmo de Euclides

mdc(m,n)

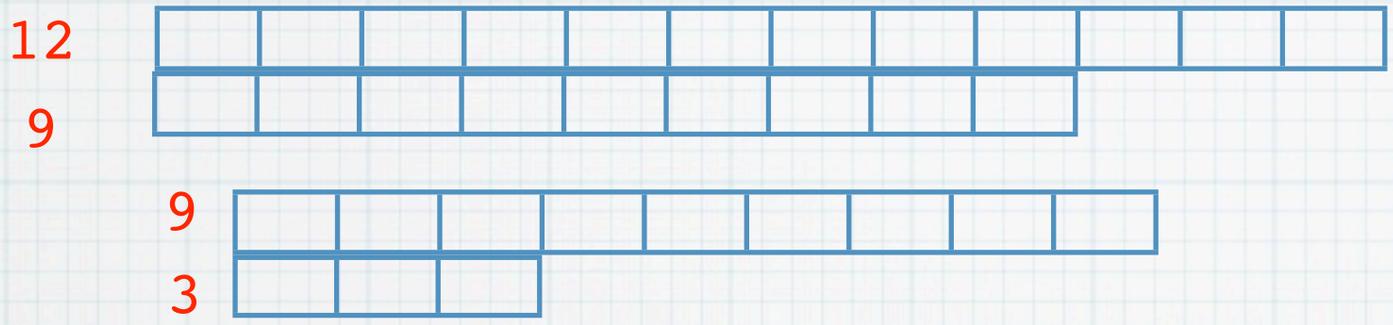


mdc(12,9)=3

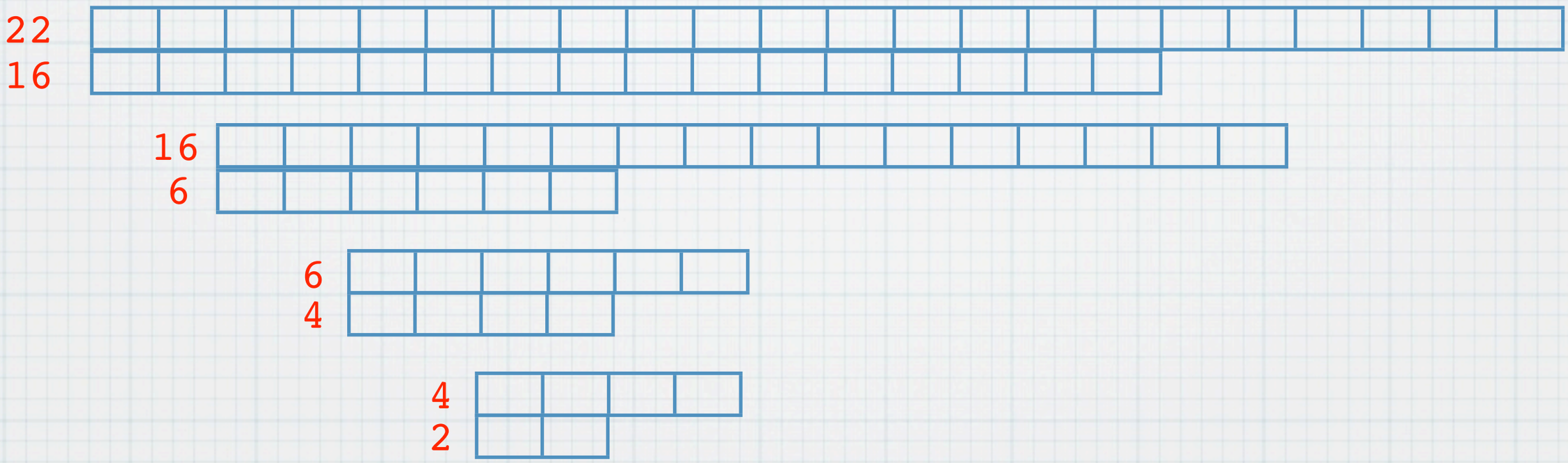


Algoritmo de Euclides

mdc(m,n)

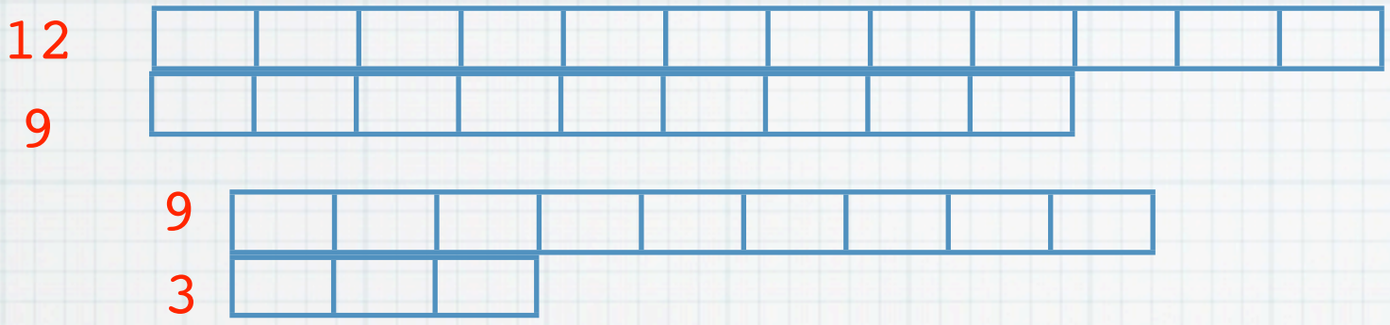


mdc(12,9)=3

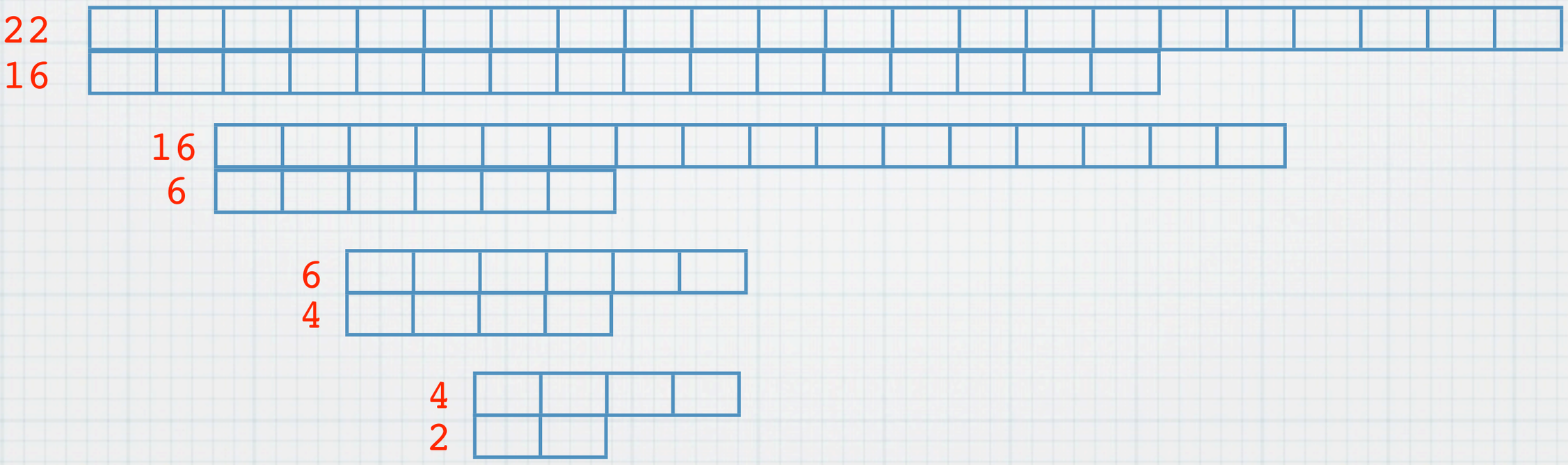


Algoritmo de Euclides

mdc(m,n)



mdc(12,9)=3



mdc(22,16)=2

Algoritmo de Euclides

Algoritmo de Euclides

* Sejam os inteiros m e n e suponhamos m<n.

Algoritmo de Euclides

* Sejam os inteiros m e n e suponhamos m<n.
* Dividir n por m.

Algoritmo de Euclides

* Sejam os inteiros m e n e suponhamos m<n.
* Dividir n por m.
* Se o resto r é zero, então m.d.c(m,n)=n.

Algoritmo de Euclides

* Sejam os inteiros `m` e `n` e suponhamos `m<n`.
* Dividir `n` por `m`.
* Se o resto `r` é zero, então `m.d.c(m,n)=n`.
* Senão dividir `m` por `r` e repetir.

Pseudo-código

```
Sejam os inteiros m e n e suponhamos
m < n.
Enquanto m > 0 fazer
    r = n % m
    n = m
    m = r
Escreve n
```

Pseudo-código

```
#include <stdio.h>
int main()
{
   int n,m,r;
   printf("Valores: ");
   scanf("%d %d",&n,&m);
   printf("O m.d.c(%d,%d)= ",n,m);
     while( m != 0 ){
        r = n%m;
        n = m;
        m = r;
     }
   printf("%d\n", n);
   return 0;
}
```

Correção do Algoritmo

Correção do Algoritmo

* Sejam `Mi` e `Ni`, `i = 0,1,...` os sucessivos valores das variáveis `m` e `n`, respectivamente.
* Pretende-se provar que se o algoritmo parar ao fim de `k` iterações, então `m.d.c(M0,N0)=Nk`.

Terminação

Terminação

* Em cada iteração, o valor de m decresce e inicialmente o valor de m é positivo.

Correção do Algoritmo

Correção do Algoritmo

* Se `k=1` então `N0%M0=0` e `N1=M0`. Isto significa que `n` é múltiplo de `m`, o `m.d.c(m,n)=m` e, portanto, o algoritmo está correcto.

Correção do Algoritmo

* Se `k=1` então `N0%M0=0` e `N1=M0`. Isto significa que `n` é múltiplo de `m`, o `m.d.c(m,n)=m` e, portanto, o algoritmo está correcto.
* Dadas duas iterações consecutivas do ciclo, `i` e `i+1`, um inteiro positivo `p` divide `Mi` e `Ni` se e só se `p` divide `M(i+1)` e `N(i+1)`

Correção do Algoritmo

* Se `k=1` então `N0%M0=0` e `N1=M0`. Isto significa que `n` é múltiplo de `m`, o `m.d.c(m,n)=m` e, portanto, o algoritmo está correcto.
* Dadas duas iterações consecutivas do ciclo, `i` e `i+1`, um inteiro positivo `p` divide `Mi` e `Ni` se e só se `p` divide `M(i+1)` e `N(i+1)`
* **Mostra a afirmação anterior!**

Correção do Algoritmo

Correção do Algoritmo

Então, o algoritmo está correcto, porque

Correção do Algoritmo

Então, o algoritmo está correcto, porque

* m.d.c(Mi,Ni)=m.d.c(M(i+1),N(i+1))

Correção do Algoritmo

Então, o algoritmo está correcto, porque

* m.d.c(Mi,Ni)=m.d.c(M(i+1),N(i+1))
* m.d.c(M(k-1),N(k-1)) = M(k-1)

Correção do Algoritmo

Então, o algoritmo está correcto, porque

* m.d.c(Mi,Ni)=m.d.c(M(i+1),N(i+1))
* m.d.c(M(k-1),N(k-1)) = M(k-1)
* m.d.c(M0,N0) = Nk