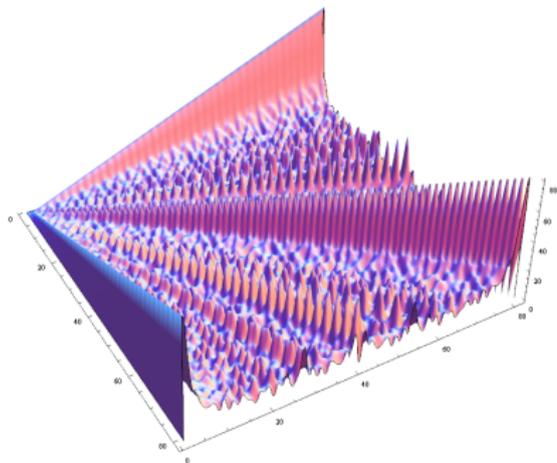


AULA 4

Hoje

- ▶ mais **análise de algoritmos**:
algoritmo de Euclides
- ▶ mais **recursão**: curvas de Hilbert

Algoritmo de Euclides



Fonte: <http://math.stackexchange.com/>

PF 2 (Exercícios) S 5.1

<http://www.ime.usp.br/~pf/algoritmos/aulas/recu.html>

<http://www.ime.usp.br/~coelho/mac0122-2014/aulas/mdc/>

Algoritmo de Euclides

O máximo divisor comum pode ser determinado através de um algoritmo de 2300 anos (cerca de 300 A.C.), o **algoritmo de Euclides**.

Para calcular o $\text{mdc}(m, n)$ o algoritmo de Euclides usa a recorrência:

$$\text{mdc}(m, 0) = m;$$

$$\text{mdc}(m, n) = \text{mdc}(n, m \% n), \text{ para } n > 0.$$

Assim, por exemplo,

$$\text{mdc}(12, 18) = \text{mdc}(18, 12) = \text{mdc}(12, 6) = \text{mdc}(6, 0) = 6.$$

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

```
/* Pre-condicao: a funcao supoe  $n > 0$  */  
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)
```

$\text{mdc}(317811,514229) = 1.$

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    0m1.692s
user    0m1.684s
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

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

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

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

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

... = ...

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

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

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

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

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 $\text{euclidesR}(514229, 317811)$ foram 27.

Consumo de tempo

Resumindo, a quantidade de tempo consumida pelo algoritmo de Euclides é, no pior caso, **proporcional** a $\lg n$.

Este desempenho é **significativamente melhor** que o desempenho do algoritmo *café com leite*, já que a função $f(n) = \lg n$ cresce **muito mais lentamente** que a função $g(n) = n$.

Consumo de tempo

n	$(\text{int}) \lg n$
4	2
5	2
6	2
10	3
64	6
100	6
128	7
1000	9
1024	10
1000000	19
1000000000	29

Conclusões

Suponha que $m > n$.

O número de chamadas recursivas da função `euclidesR` é $\leq 2(\lg n) - 1$.

No pior caso, o consumo de tempo da função `euclidesR` é proporcional a $\lg n$.

Conclusões

Suponha que $m > n$.

O consumo de tempo da função `euclidesR` é $O(\lg n)$.

Para que o consumo de tempo da função `euclidesR` dobre é necessário que o valor de n seja elevado ao quadrado.

Euclides e Fibonacci

Demonstre por indução em k que:

Se $m > n \geq 0$ e se a chamada $\text{euclidesR}(m,n)$ faz $k \geq 1$ chamadas recursivas, então

$$m \geq \text{fibonacci}(k + 2) \text{ e } n \geq \text{fibonacci}(k + 1).$$

Curvas de Hilbert



Fonte: <http://momath.org/home/math-monday-03-22-10>

Niklaus Wirth, *Algorithms and Data Structures*
Prentice Hall, 1986.

http://en.wikipedia.org/wiki/Hilbert_curve

Curvas de Hilbert

As curvas a seguir seguem um certo **padrão regular** e podem ser desenhadas na tela sobre o controle de um programa.

O objetivo é descobrir o **esquema de recursão** para construir tais curvas.

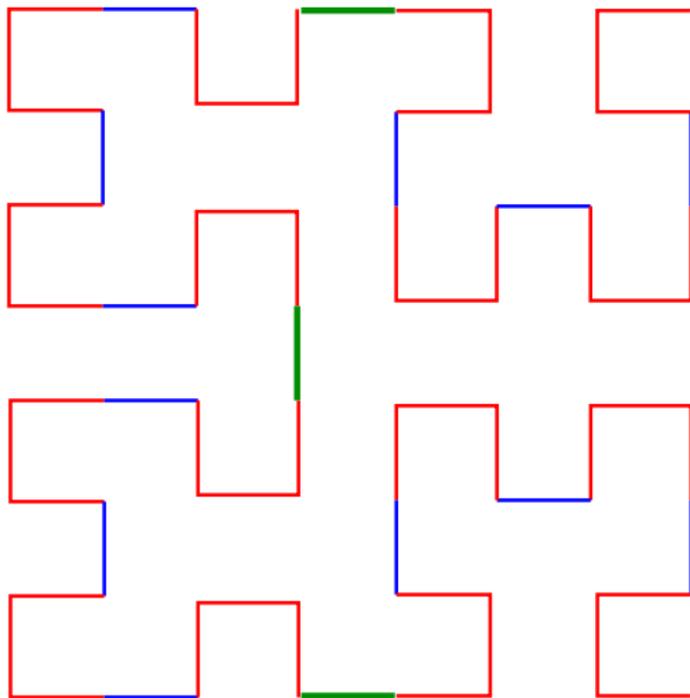
Estes padrões serão chamados de $H_0, H_1, H_2 \dots$

Cada H_i denomina a **curva de Hilbert** de **ordem i** , em homenagem a seu inventor, o matemático *David Hilbert*.

H_1



H_3



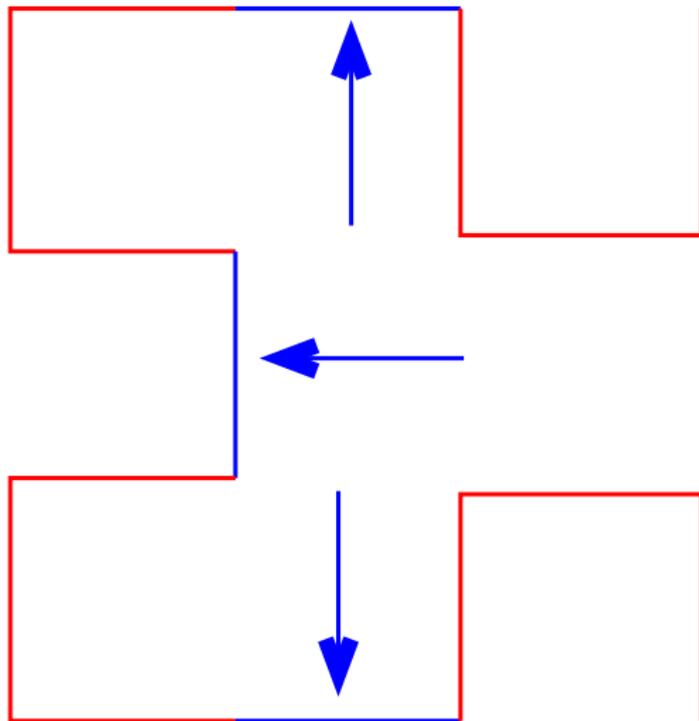
Padrão

As figuras mostram que H_{i+1} é obtida pela composição de 4 instâncias de H_i de metade do tamanho e com a rotação apropriada, ligadas entre si por meio de 3 linhas de conexão.

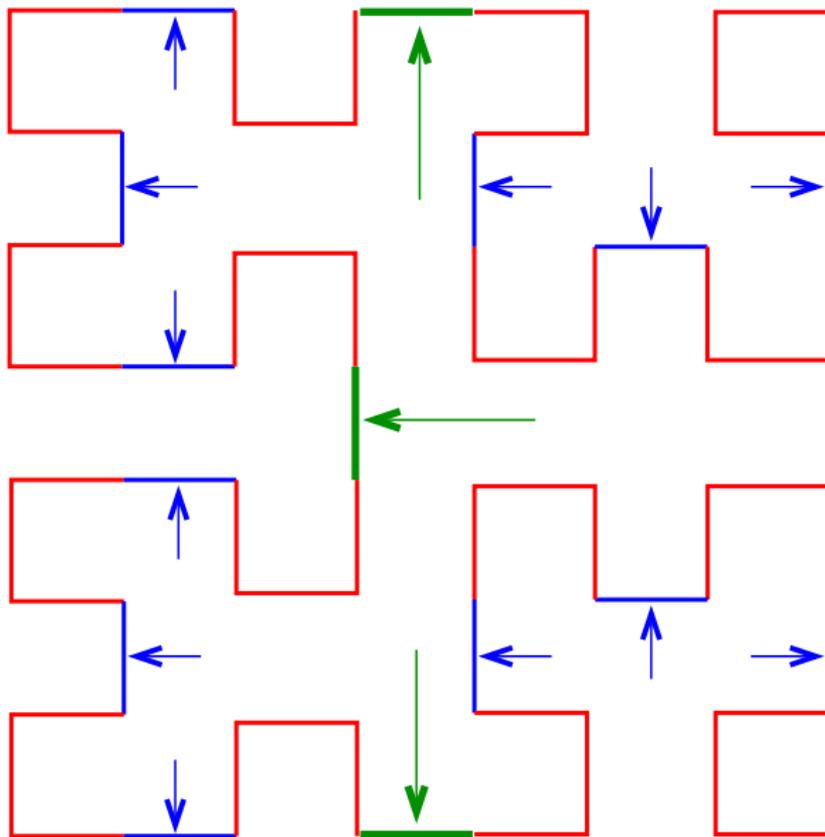
Por exemplo:

- ▶ H_1 é formada por 4 H_0 (vazio) conectados por 3 linhas.
- ▶ H_2 é formada por 4 H_1 conectados por 3 linhas
- ▶ H_3 é formada por 4 H_2 conectados por 3 linhas

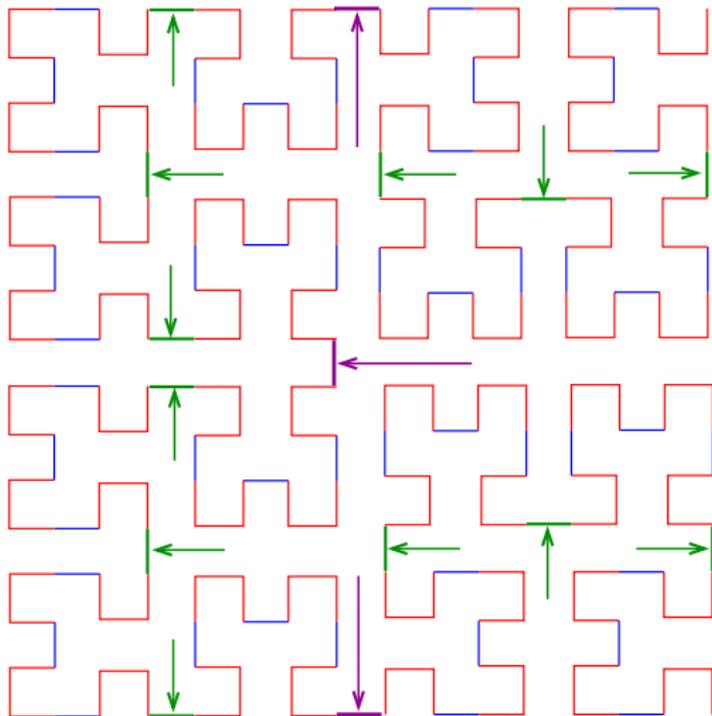
H₂



H_3



H_4



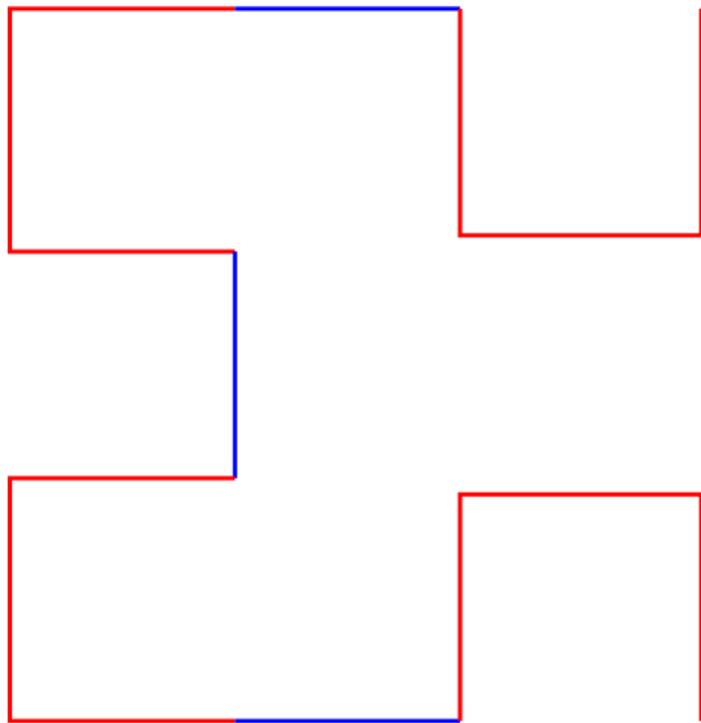
Partes da curva

Para ilustrar, denotaremos as quatro possíveis instâncias por **A**, **B**, **C** e **D**:

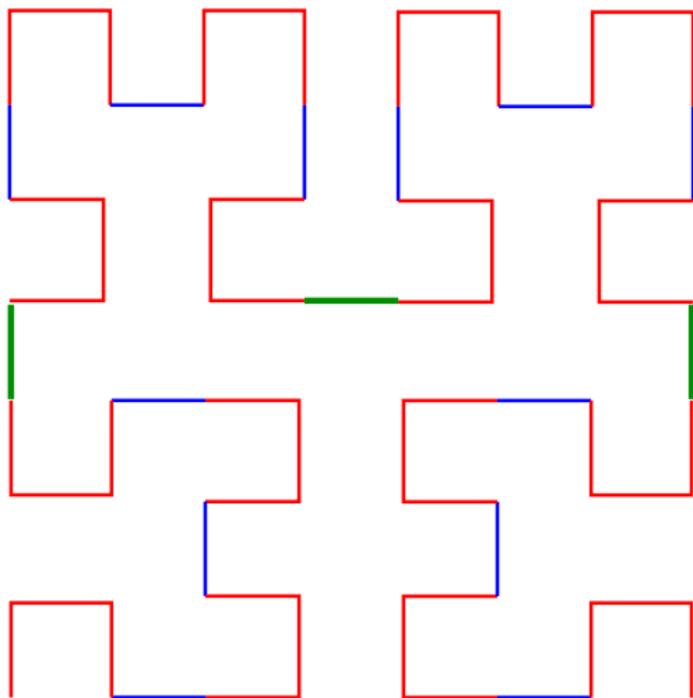
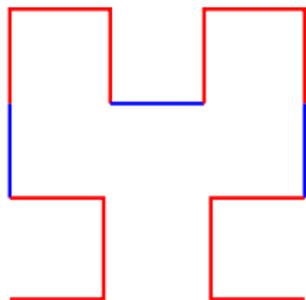
- ▶ **A** será o padrão que tem a “abertura” para **direita**;
- ▶ **B** será o padrão que tem a “abertura” para **baixo**;
- ▶ **C** será o padrão que tem a “abertura” para **esquerda**; e
- ▶ **D** será o padrão que tem a “abertura” para **cima**.

Representaremos a chamada da função que desenha as interconexões por meio das setas \uparrow , \downarrow , \leftarrow , \rightarrow .

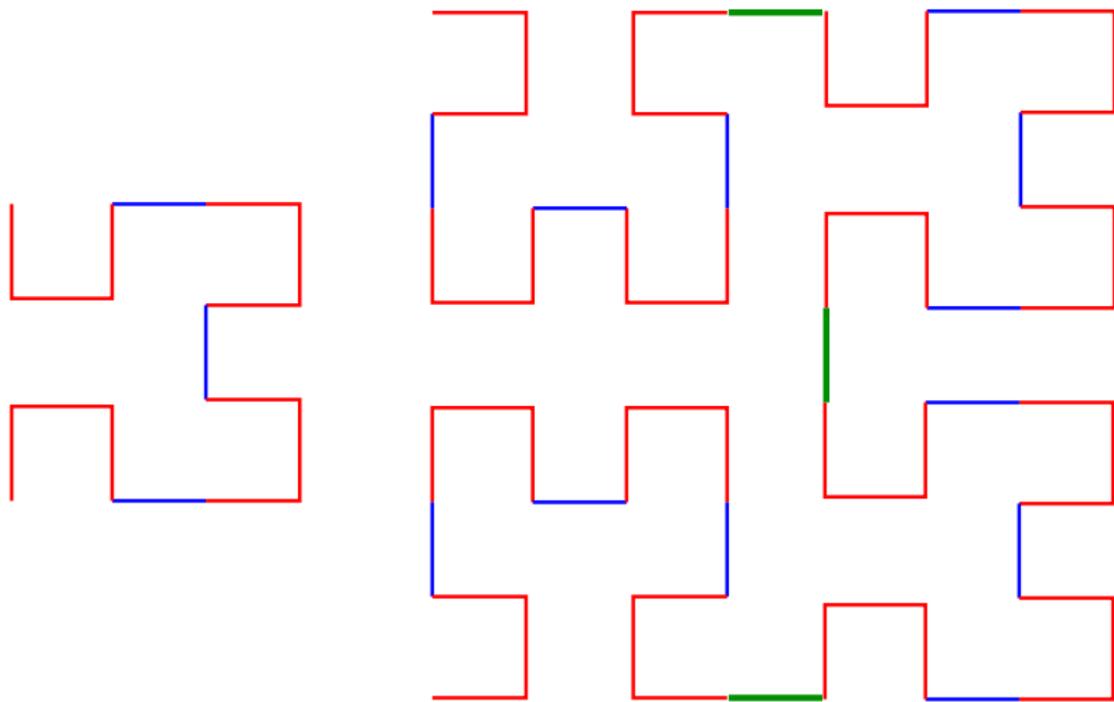
$A_1 \in A_2$



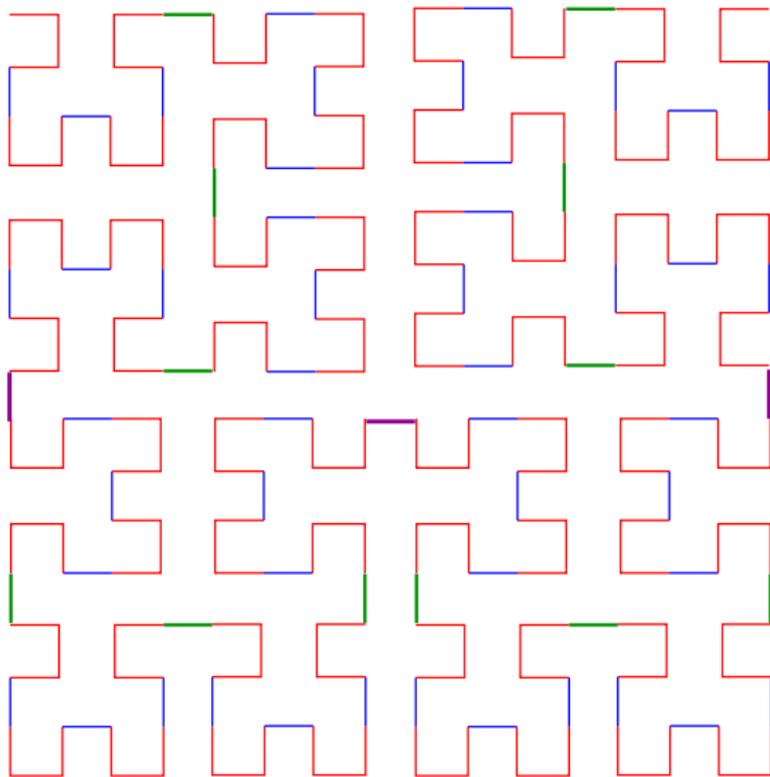
B_2 e B_3



C_2 e C_3



D_4



Esquema recursivo

Assim, surge o seguinte esquema recursivo:

$$\begin{array}{l} A_k : D_{k-1} \leftarrow A_{k-1} \downarrow A_{k-1} \rightarrow B_{k-1} \\ B_k : C_{k-1} \uparrow B_{k-1} \rightarrow B_{k-1} \downarrow C_{k-1} \\ C_k : B_{k-1} \rightarrow C_{k-1} \uparrow C_{k-1} \leftarrow D_{k-1} \\ D_k : A_{k-1} \downarrow D_{k-1} \leftarrow D_{k-1} \uparrow C_{k-1} \end{array}$$

Para desenhar os segmentos utilizaremos a chamada de uma função

`linha(x,y,direcao,comprimento)`

que “**move um pincel**” da posição (x,y) em uma dada **direcao** por um certo **comprimento**.

```
typedef enum {DIREITA, ESQUERDA, CIMA, BAIXO} Direcao;
void linha(int *x, int *y, Direcao direcao,
           int comprimento) {
    switch (direcao) {
        case DIREITA : *x = *x + comprimento;
                       break;
        case ESQUERDA : *x = *x - comprimento;
                       break;
        case CIMA : *y = *y + comprimento;
                   break;
        case BAIXO : *y = *y - comprimento;
                   break;
    }
    desenhaLinha(*x, *y);
}
```

A_k

```
void
a(int k, int *x, int *y, int comprimento)
{
    if (k > 0) {
        d(k-1, x, y, comprimento);
        linha(x, y, ESQUERDA, comprimento);
        a(k-1, x, y, comprimento);
        linha(x, y, BAIXO, comprimento);
        a(k-1, x, y, comprimento);
        linha(x, y, DIREITA, comprimento);
        b(k-1, x, y, comprimento);
    }
}
```

B_k

```
void
b(int k, int *x, int *y, int comprimento)
{
    if (k > 0) {
        c(k-1, x, y, comprimento);
        linha(x, y, CIMA, comprimento);
        b(k-1, x, y, comprimento);
        linha(x, y, DIREITA, comprimento);
        b(k-1, x, y, comprimento);
        linha(x, y, BAIXO, comprimento);
        a(k-1, x, y, comprimento);
    }
}
```

C_k

```
void
c(int k, int *x, int *y, int comprimento)
{
    if (k > 0) {
        b(k-1, x, y, comprimento);
        linha(x, y, DIREITA, comprimento);
        c(k-1, x, y, comprimento);
        linha(x, y, CIMA, comprimento);
        c(k-1, x, y, comprimento);
        linha(x, y, ESQUERDA, comprimento);
        d(k-1, x, y, comprimento);
    }
}
```

D_k

```
void
d(int k, int *x, int *y, int comprimento)
{
    if (k > 0) {
        a(k-1, x, y, comprimento);
        linha(x, y, BAIXO, comprimento);
        d(k-1, x, y, comprimento);
        linha(x, y, ESQUERDA, comprimento);
        d(k-1, x, y, comprimento);
        linha(x, y, CIMA, comprimento);
        c(k-1, x, y, comprimento);
    }
}
```