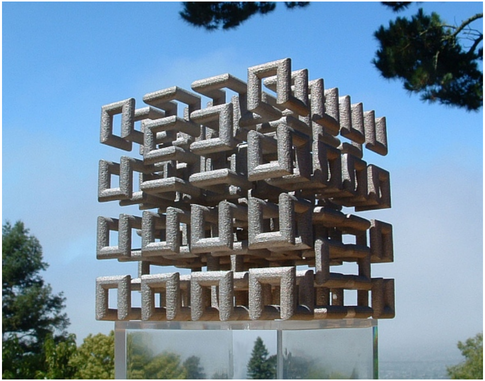
Curvas de Hilbert



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.

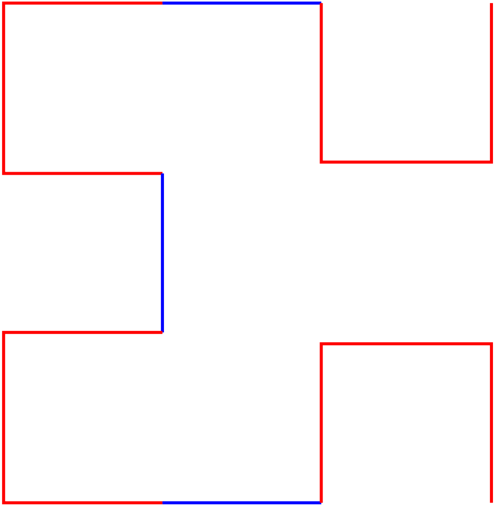
O objetivo é descobrir o esquema de recursão para construir tais curvas.

Estes padrões serão chamados de $\mathtt{H}_0, \mathtt{H}_1, \mathtt{H}_2 \ldots$.

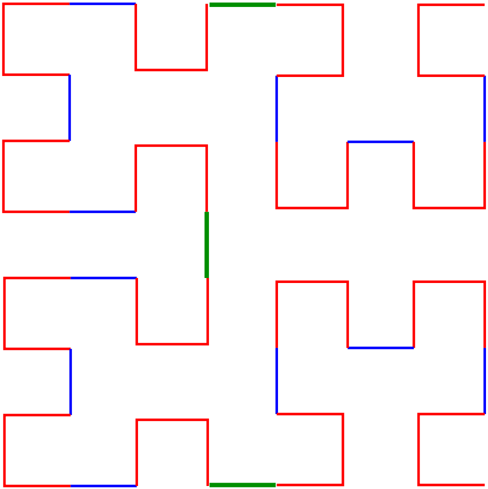
Cada $\mathtt{H}_\mathtt{i}$ denomina a **curva de Hilbert** de ordem $\mathtt{i}$, em homenagem a seu inventor, o matemático *David Hilbert*.

H_1

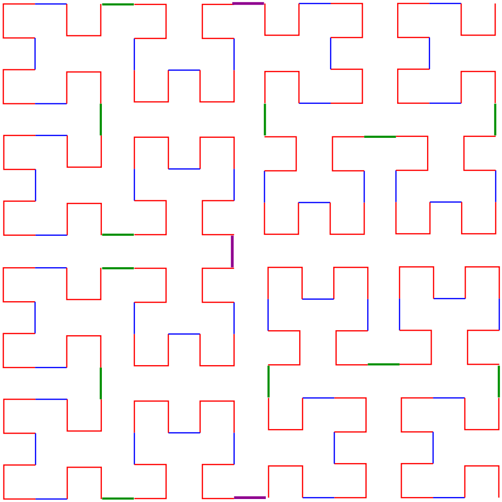
H_2



H_3



H_4



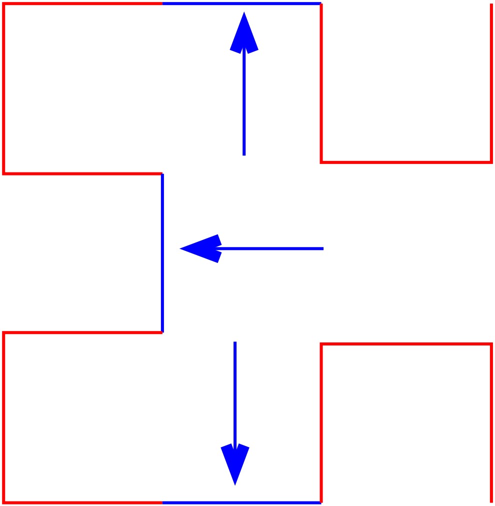
Padrão

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

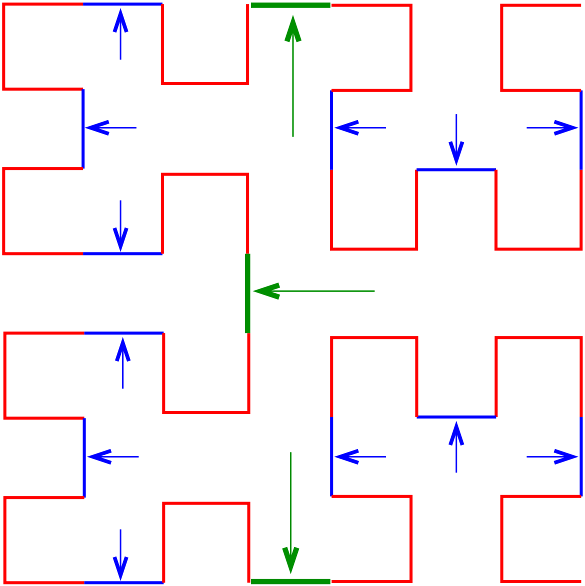
Por exemplo:

- $\mathtt{H}_1$ é formada por 4 $\mathtt{H}_0$ (vazio) conectados por 3 linhas.
- $\mathtt{H}_2$ é formada por 4 $\mathtt{H}_1$ conectados por 3 linhas
- $\mathtt{H}_3$ é formada por 4 $\mathtt{H}_2$ conectados por 3 linhas

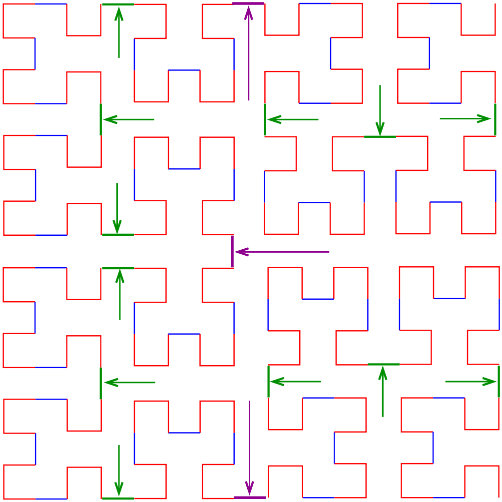
H_2



H_3



H_4



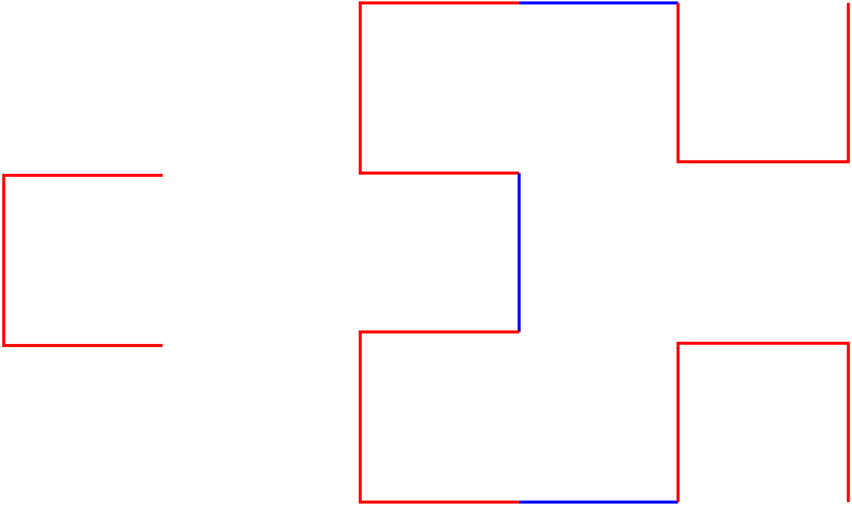
Partes da curva

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

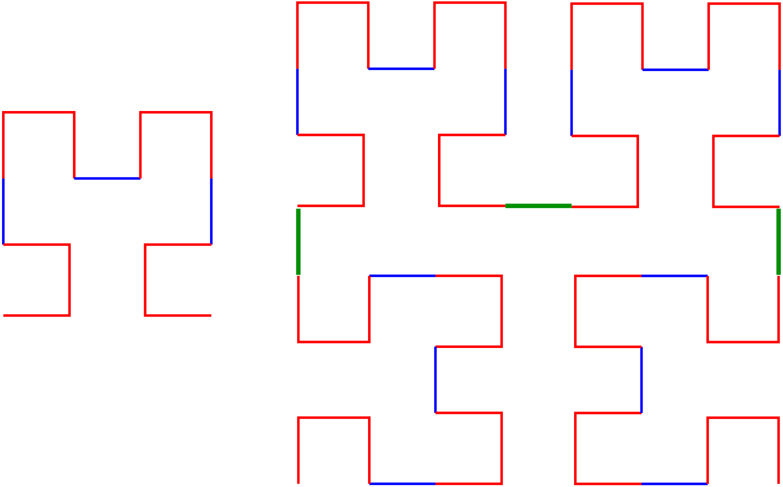
- A será o padrão que tem a "abertura" para direita;
- B será o padrão que tem a "abertura" para baixo;
- C será o padrão que tem a "abertura" para esquerda; e
- D será o padrão que tem a "abertura" para cima.

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

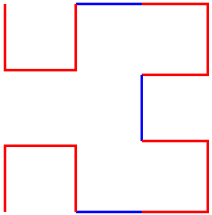
A_1 e A_2

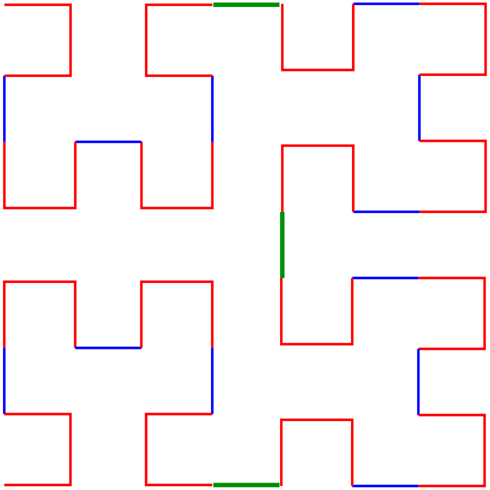


B_2 e B_3

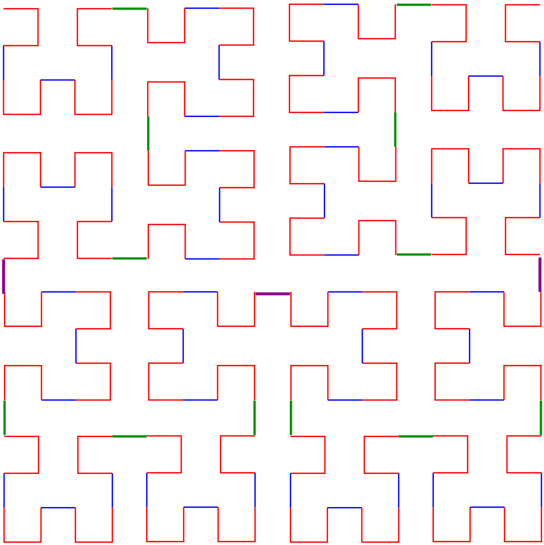


C_2 e C_3





D_4



Esquema recursivo

Assim, surge o seguinte esquema recursivo:

$$
\begin{array}{lllllllll}
\mathtt{A}_{\mathtt{k}} & : & \mathtt{D}_{\mathtt{k}-1} & \leftarrow & \mathtt{A}_{\mathtt{k}-1} & \downarrow & \mathtt{A}_{\mathtt{k}-1} & \rightarrow & \mathtt{B}_{\mathtt{k}-1} \\
\mathtt{B}_{\mathtt{k}} & : & \mathtt{C}_{\mathtt{k}-1} & \uparrow & \mathtt{B}_{\mathtt{k}-1} & \rightarrow & \mathtt{B}_{\mathtt{k}-1} & \downarrow & \mathtt{C}_{\mathtt{k}-1} \\
\mathtt{C}_{\mathtt{k}} & : & \mathtt{B}_{\mathtt{k}-1} & \rightarrow & \mathtt{C}_{\mathtt{k}-1} & \uparrow & \mathtt{C}_{\mathtt{k}-1} & \leftarrow & \mathtt{D}_{\mathtt{k}-1} \\
\mathtt{D}_{\mathtt{k}} & : & \mathtt{A}_{\mathtt{k}-1} & \downarrow & \mathtt{D}_{\mathtt{k}-1} & \leftarrow & \mathtt{D}_{\mathtt{k}-1} & \uparrow & \mathtt{C}_{\mathtt{k}-1}
\end{array}
$$

Para desenhar os segmentos utilizaremos a chamada de uma função

```
move(pincel,direcao,comprimento)
```

que "move um pincel" da posição ($\mathtt{x}$,$\mathtt{y}$) em uma dada `direcao` por um certo `comprimento`.

move()

```
def move(pincel, direcao, comprimento):
    # pegue a posição do pincel
    x, y = pincel.pos()
    # calcule nova posição
    if direcao== DIREITA:
        x = x + comprimento
    elif direcao == ESQUERDA:
        x = x - comprimento
    elif direcao == CIMA:
        y = y + comprimento
    elif direcao == BAIXO:
        y = y - comprimento
    pincel.goto(x,y)
```

A_k

```
def a(k, pincel, comprimento):
    '''(int, Turtle, int) -> None
    Desenha a curva A_k a partir da
posição
    do pincel.
    '''
    if k > 0:
        d(k-1, pincel, comprimento)
        move(pincel, ESQUERDA, comprimento)
        a(k-1, pincel, comprimento)
        move(pincel, BAIXO, comprimento)
        a(k-1, pincel, comprimento)
        move(pincel, DIREITA, comprimento)
        b(k-1, pincel, comprimento)
```

B_k

```
def b(k, pincel, comprimento):
    '''(int, Turtle, int) -> None
    Desenha a curva B_k a partir da
posição
    do pincel.
    '''
    if k > 0:
        c(k-1, pincel, comprimento)
        move(pincel, CIMA, comprimento)
        b(k-1, pincel, comprimento)
        move(pincel, DIREITA, comprimento)
        b(k-1, pincel, comprimento)
        move(pincel, BAIXO, comprimento)
        a(k-1, pincel, comprimento)
```

C_k

```
def c(k, pincel, comprimento):
    '''(int, Turtle, int) -> None
    Desenha a curva C_k a partir da
posição
    do pincel.
    '''
    if k > 0:
        b(k-1, pincel, comprimento)
        move(pincel, DIREITA, comprimento)
        c(k-1, pincel, comprimento)
        move(pincel, CIMA, comprimento)
        c(k-1, pincel, comprimento)
        move(pincel, ESQUERDA, comprimento)
        d(k-1, pincel, comprimento)
```

D_k

```
def d(k, pincel, comprimento):
    '''(int, Turtle, int) -> None
    Desenha a curva D_k a partir da
posição
    do pincel.
    '''
    if k > 0:
        a(k-1, pincel, comprimento)
        move(pincel, BAIXO, comprimento)
        d(k-1, pincel, comprimento)
        move(pincel, ESQUERDA, comprimento)
        d(k-1, pincel, comprimento)
        move(pincel, CIMA, comprimento)
        c(k-1, pincel, comprimento)
```