Melhores momentos

AULA 2

Conceitos discutidos

- um pouco mais de recursão
- um pouco de análise experimental de algoritmos
- um pouco de análise de algoritmos

Desempenho de fibonacciR

Algoritmo recursivo para F_n:

```
long fibonacciR(int n) {
   if (n == 0) return 0;
   if (n == 1) return 1;
   return fibonacciR(n-1) +
          fibonacciR(n-2);
}
```

Resolve subproblemas muitas vezes

```
fibonacciR(8)
  fibonacciR(7)
    fibonacciR(6)
      fibonacciR(5)
        fibonacciR(4)
          fibonacciR(3)
            fibonacciR(2)
              fibonacciR(1)
              fibonacciR(0)
            fibonacciR(1)
          fibonacciR(2)
            fibonacciR(1)
            fibonacciR(0)
        fibonacciR(3)
          fibonacciR(2)
            fibonacciR(1)
            fibonacciR(0)
          fibonacciR(1)
      fibonacciR(4)
        fibonacciR(3)
          fibonacciR(2)
            fibonacciR(1)
            fibonacciR(0)
          fibonacciR(1)
        fibonacciR(2)
          fibonacciR(1)
          fibonacciR(0)
```

```
        fibonacciR(1)
      fibonacciR(2)
        fibonacciR(1)
        fibonacciR(0)
    fibonacciR(5)
      fibonacciR(4)
        fibonacciR(3)
          fibonacciR(2)
            fibonacciR(1)
            fibonacciR(0)
          fibonacciR(1)
        fibonacciR(2)
          fibonacciR(1)
          fibonacciR(0)
      fibonacciR(3)
        fibonacciR(2)
          fibonacciR(1)
          fibonacciR(0)
        fibonacciR(1)
  fibonacciR(6)
    fibonacciR(5)
      fibonacciR(4)
        fibonacciR(3)
```

```
          fibonacciR(2)
            fibonacciR(1)
            fibonacciR(0)
          fibonacciR(1)
        fibonacciR(2)
          fibonacciR(1)
          fibonacciR(0)
      fibonacciR(3)
        fibonacciR(2)
          fibonacciR(1)
          fibonacciR(0)
        fibonacciR(1)
    fibonacciR(4)
      fibonacciR(3)
        fibonacciR(2)
          fibonacciR(1)
          fibonacciR(0)
        fibonacciR(1)
      fibonacciR(2)
        fibonacciR(1)
        fibonacciR(0)
fibonacci(8) = 21.
```

Comparação experimental

```
meu_prompt> time ./fibonacciI 45
fibonacci(45) = 1134903170
real                    0m0.003s
user                    0m0.001s
sys                     0m0.001s

meu_prompt> time ./fibonacciR 45
fibonacci(45) = 1134903170
real                    0m8.486s
user                    0m8.459s
sys                     0m0.008s
```

Conclusões

Devemos **evitar** resolver o
mesmo subproblema várias vezes.

O número de chamadas recursivas
feitas por `fibonacciR(n)` é

$$2 \times (\texttt{F}(\texttt{n} + 1) - 1)\,.$$

Mais conclusões

O consumo de tempo da chamada `fibonacciR(n)` é *proporcional a*

$$2 \times \mathtt{F}(\mathtt{n}) - 2\,.$$

O consumo de tempo da chamada `fibonacciR(n)` é *exponencial* pois

$$\mathtt{F}(\mathtt{n}) \geq \left(\frac{3}{2}\right)^{\mathtt{n}-2} \quad \text{para } \mathtt{n} \geq 2.$$

Pausa para nossos comerciais

- Plantão de dúvidas: Francisco e Lorenzo

  Horário: terças das 15h às 16h

  Link: https://meet.google.com/ztw-onpf-hze

- Grupo MaratonUSP:

  http://www.ime.usp.br/~maratona/

- Grupo PoliBits:

  Responsável para treinar para a OBI ali na poli

  Horário: sextas às 17h

  Onde: sala GD-06 do prédio da Elétrica

Hoje

- mais recursão
- mais análise experimental de algoritmos
- um pouco de correção de algoritmos: invariantes
- um pouco de análise de algoritmos:
  "consumo de tempo proporcional a" e

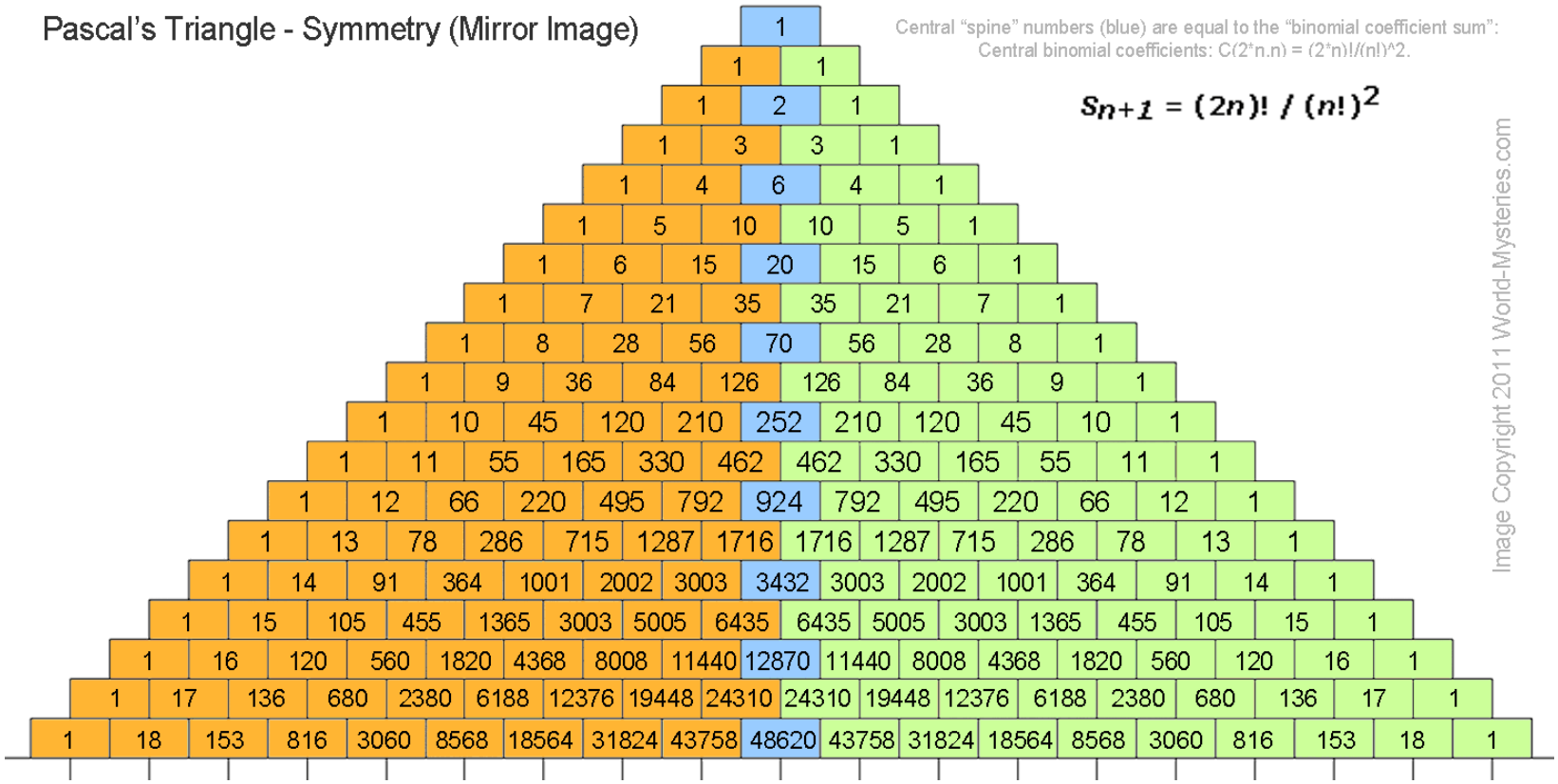
  notação assintótica

Mais recursão ainda

PF 2.1, 2.2, 2.3 S 5.1

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

Números binomiais



Fonte: http://blog.world-mysteries.com/

PF 2 (Exercícios)

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

Binomial recursivo

Regra de Pascal

$$\binom{n}{k} = \begin{cases} 0, & \text{quando } n = 0 \text{ e } k > 0, \\ 1, & \text{quando } n \geq 0 \text{ e } k = 0, \\ \binom{n-1}{k} + \binom{n-1}{k-1}, & \text{quando } n, k > 0. \end{cases}$$

Binomial

	0	1	2	3	4	5	6	7	8	...	k
0	1	0	0	0	0	0	0	0	0	...	
1	1	1	0	0	0	0	0	0	0	...	
2	1	2	1	0	0	0	0	0	0	...	
3	1	3	3	1	0	0	0	0	0	...	
4	1	4	6	4	1	0	0	0	0	...	
5	1	5	10	10	5	1	0	0	0	...	
6	1	6	15	20	15	6	1	0	0	...	
7	1	7	21	35	35	21	7	1	0	...	
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋱	
n											

Binomial

	0	1	2	3	4	5	6	7	8	...	k
0	1	0	0	0	0	0	0	0	0	...	
1	1	1	0	0	0	0	0	0	0	...	
2	1	2	1	0	0	0	0	0	0	...	
3	1	3	3	1	0	0	0	0	0	...	
4	1	4	6	4	1	0	0	0	0	...	
5	1	5	10	10	5	1	0	0	0	...	
6	1	6	15	20	15	6	1	0	0	...	
7	1	7	21	35	35	21	7	1	0	...	
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋱	
n											

Mais eficiente . . .

Versão anterior da regra de Pascal:

$$\binom{n}{k} = \begin{cases} 0, & \text{quando } n = 0 \text{ e } k > 0, \\ 1, & \text{quando } n \geq 0 \text{ e } k = 0, \\ \binom{n-1}{k} + \binom{n-1}{k-1}, & \text{quando } n, k > 0. \end{cases}$$

Outro jeito de definir a regra de Pascal:

$$\binom{n}{k} = \begin{cases} 0, & \text{quando } n < k, \\ 1, & \text{quando } n = k \text{ ou } k = 0, \\ \binom{n-1}{k} + \binom{n-1}{k-1}, & \text{quando } n, k > 0. \end{cases}$$

Mais eficiente . . .

	0	1	2	3	4	5	6	7	8	. . .	k
0	1	0	0	0	0	0	0	0	0	. . .	
1	1	1	0	0	0	0	0	0	0	. . .	
2	1	2	1	0	0	0	0	0	0	. . .	
3	1	3	3	1	0	0	0	0	0	. . .	
4	1	4	6	4	1	0	0	0	0	. . .	
5	1	5	10	10	5	1	0	0	0	. . .	
6	1	6	15	20	15	6	1	0	0	. . .	
7	1	7	21	35	35	21	7	1	0	. . .	
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋱	
n											

Mais eficiente . . .

$$\binom{n}{k} = \begin{cases} 0, & \text{quando } n < k, \\ 1, & \text{quando } n = k \text{ ou } k = 0, \\ \binom{n-1}{k} + \binom{n-1}{k-1}, & \text{quando } n, k > 0. \end{cases}$$

```
 long binomialR1(int n, int k) {
1   if (n < k) return 0;
2   if (n == k || k == 0) return 1;
3   return binomialR1(n-1, k) +
4          binomialR1(n-1, k-1);
 }
```

binomialR1(3,2)

```
binomialR1(3,2)
  binomialR1(2,2)
  binomialR1(2,1)
    binomialR1(1,1)
    binomialR1(1,0)
binom(3,2)=3.
```

binomialR1(5,4)

```
binomialR1(5,4)
  binomialR1(4,4)
  binomialR1(4,3)
    binomialR1(3,3)
    binomialR1(3,2)
      binomialR1(2,2)
      binomialR1(2,1)
        binomialR1(1,1)
        binomialR1(1,0)
binom(5,4)=5.
```

Binomial iterativo

```
long binomialI(int n, int k) {
   int i, j, bin[MAX][MAX];

   for (j = 1; j <= k; j++) bin[0][j] = 0;
   for (i = 0; i <= n; i++) bin[i][0] = 1;

   for (i = 1; i <= n; i++)
      for (j = 1; j <= k; j++)
         bin[i][j] = bin[i-1][j] +
                     bin[i-1][j-1];

   return bin[n][k];
}
```

E agora? Qual é mais eficiente?

```
meu_prompt> time ./binomialI 30 20
binom(30,20)=30045015
real                                0m0.003s
user                                0m0.001s
sys                                 0m0.001s

meu_prompt> time ./binomialR1 30 20
binom(30,20)=30045015
real                                0m0.191s
user                                0m0.188s
sys                                 0m0.002s
```

E agora? Qual é mais eficiente?

```
meu_prompt> time ./binomialI 40 30
binom(40,30)=847660528
real                                0m0.003s
user                                0m0.001s
sys                                 0m0.001s

meu_prompt> time ./binomialR1 40 30
binom(40,30)=847660528
real                                0m5.519s
user                                0m5.433s
sys                                 0m0.009s
```

Resolve subproblemas muitas vezes?

```
binomialR1(6,4)
  binomialR1(5,4)
    binomialR1(4,4)
    binomialR1(4,3)
      binomialR1(3,3)
      binomialR1(3,2)
        binomialR1(2,2)
        binomialR1(2,1)
          binomialR1(1,1)
          binomialR1(1,0)
  binomialR1(5,3)
    binomialR1(4,3)
      binomialR1(3,3)
      binomialR1(3,2)
        binomialR1(2,2)
```

```
        binomialR1(2,1)
          binomialR1(1,1)
          binomialR1(1,0)
    binomialR1(4,2)
      binomialR1(3,2)
        binomialR1(2,2)
        binomialR1(2,1)
          binomialR1(1,1)
          binomialR1(1,0)
      binomialR1(3,1)
        binomialR1(2,1)
          binomialR1(1,1)
          binomialR1(1,0)
        binomialR1(2,0)
binom(6,4)=15.
```

Sim!

Resolve subproblemas muitas vezes?

```
binomialR1(7,4)
  binomialR1(6,4)
    binomialR1(5,4)
      binomialR1(4,4)
      binomialR1(4,3)
        binomialR1(3,3)
        binomialR1(3,2)
          binomialR1(2,2)
          binomialR1(2,1)
            binomialR1(1,1)
            binomialR1(1,0)
    binomialR1(5,3)
      binomialR1(4,3)
        binomialR1(3,3)
        binomialR1(3,2)
          binomialR1(2,2)
          binomialR1(2,1)
            binomialR1(1,1)
            binomialR1(1,0)
      binomialR1(4,2)
        binomialR1(3,2)
          binomialR1(2,2)
          binomialR1(2,1)
            binomialR1(1,1)
```

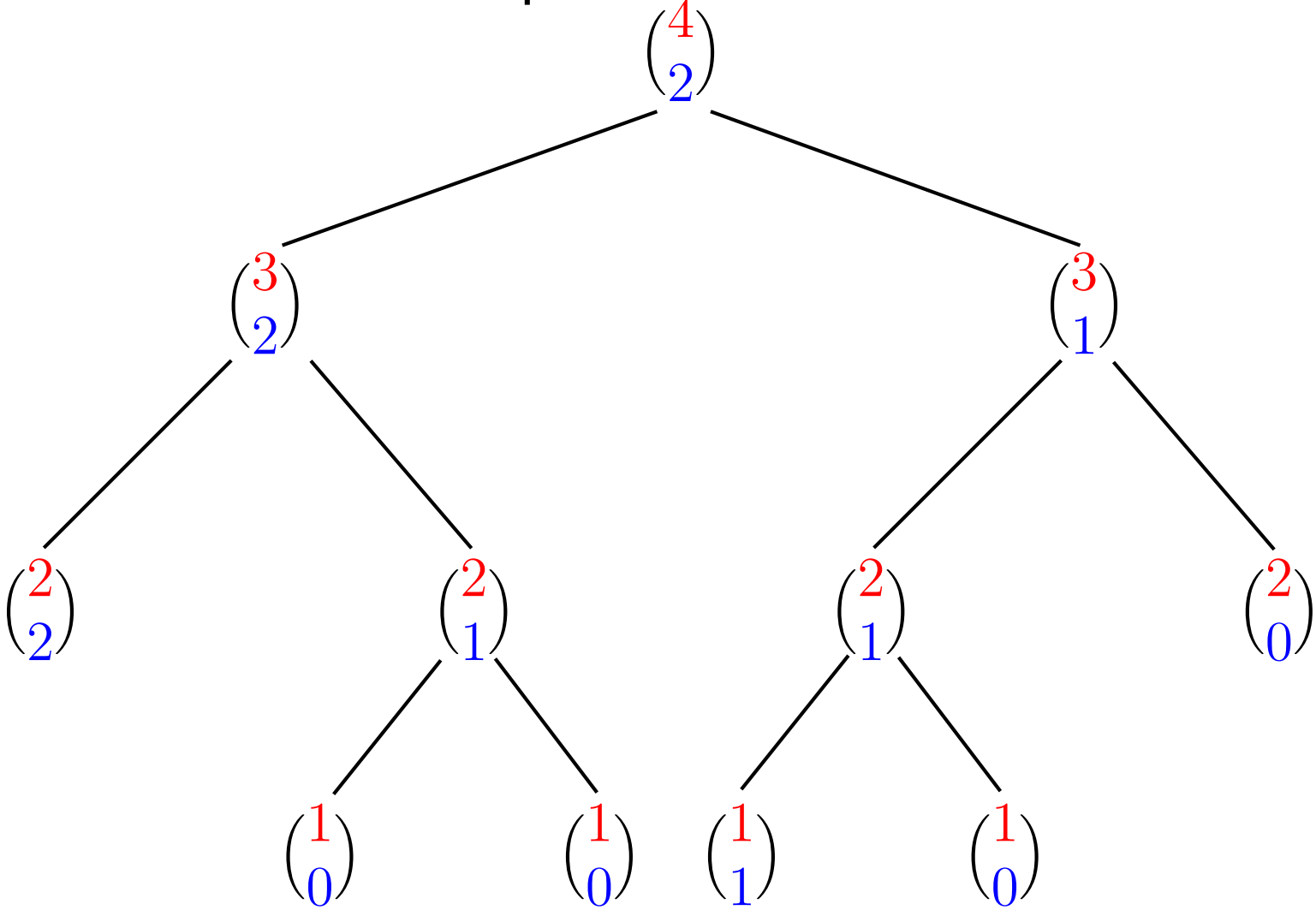
```
            binomialR1(1,0)
        binomialR1(3,1)
          binomialR1(2,1)
            binomialR1(1,1)
            binomialR1(1,0)
          binomialR1(2,0)
  binomialR1(6,3)
    binomialR1(5,3)
      binomialR1(4,3)
        binomialR1(3,3)
        binomialR1(3,2)
          binomialR1(2,2)
          binomialR1(2,1)
            binomialR1(1,1)
            binomialR1(1,0)
      binomialR1(4,2)
        binomialR1(3,2)
          binomialR1(2,2)
          binomialR1(2,1)
            binomialR1(1,1)
            binomialR1(1,0)
        binomialR1(3,1)
          binomialR1(2,1)
            binomialR1(1,1)
```

```
              binomialR1(1,0)
            binomialR1(2,0)
      binomialR1(5,2)
        binomialR1(4,2)
          binomialR1(3,2)
            binomialR1(2,2)
            binomialR1(2,1)
              binomialR1(1,1)
              binomialR1(1,0)
          binomialR1(3,1)
            binomialR1(2,1)
              binomialR1(1,1)
              binomialR1(1,0)
            binomialR1(2,0)
        binomialR1(4,1)
          binomialR1(3,1)
            binomialR1(2,1)
              binomialR1(1,1)
              binomialR1(1,0)
            binomialR1(2,0)
          binomialR1(3,0)
binom(7,4)=35.
```

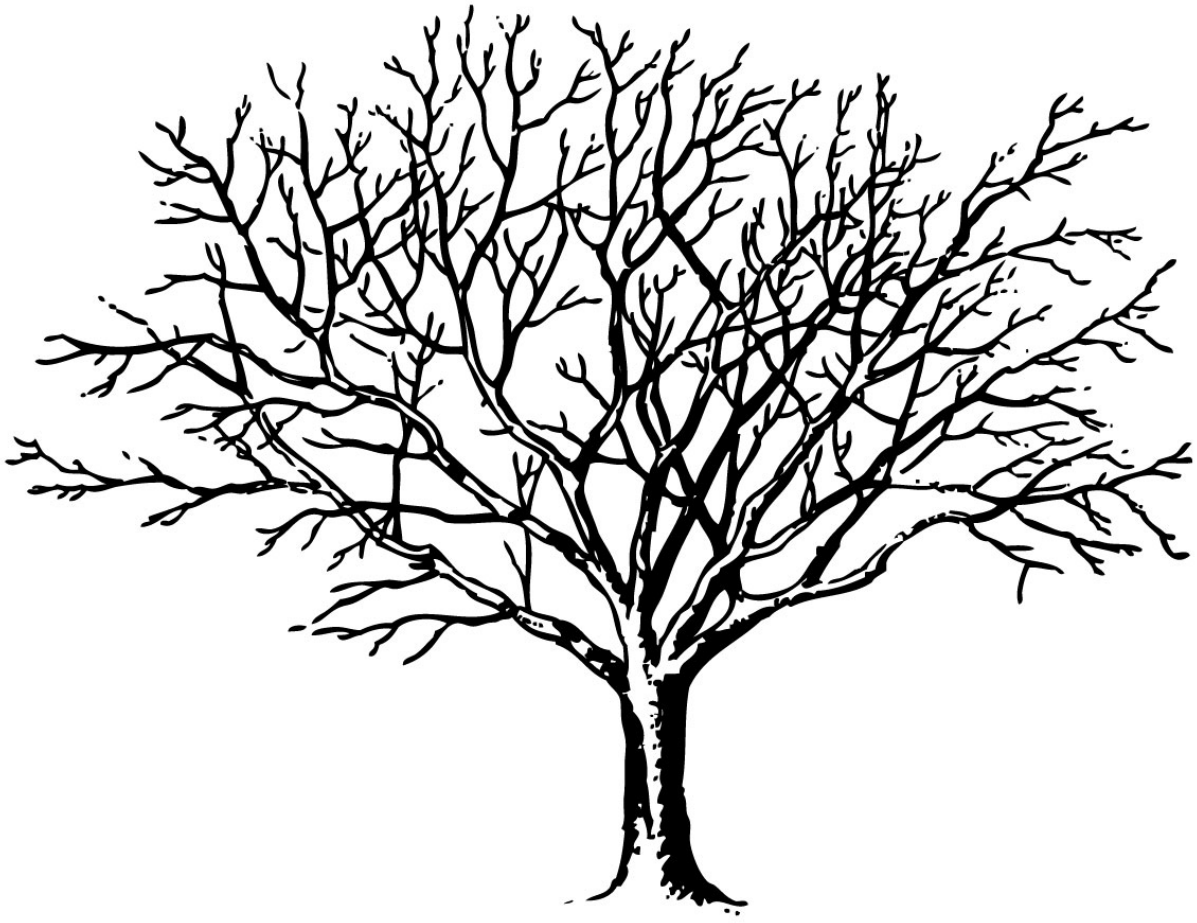
Sim!

Árvore da recursão

`binomialR1` resolve subproblemas muitas vezes.



Árvore



Fonte: http://tfhoa.com/treework

Desempenho de binomialR1

Quantas chamadas recursivas faz a função binomialR1?

```
 long binomialR1(int n, int k) {
1   if (n < k) return 0;
2   if (n == k || k == 0) return 1;
3   return binomialR1(n-1, k)
4          + binomialR1(n-1, k-1);
 }
```

Faz o dobro do número de adições.

Vamos calcular o número de adições
feitas pela chamada binomialR1(n,k).

Número de adições

Seja `T(n,k)` o número de adições
feitas pela chamada `binomialR1(n,k)`.

```
 long binomialR1(int n, int k) {
1   if (n < k) return 0;
2   if (n == k || k == 0) return 1;
3   return binomialR1(n-1, k)
4            + binomialR1(n-1, k-1);
 }
```

Número de adições

Seja $\texttt{T}(\texttt{n}, \texttt{k})$ o número de adições
feitas pela chamada $\texttt{binomialR1}(\texttt{n},\texttt{k})$.

linha	número de adições
1	$= 0$
2	$= 0$
3	$= \texttt{T}(\texttt{n}-1,\texttt{k})$
4	$= \texttt{T}(\texttt{n}-1,\texttt{k}-1) + 1$
$\texttt{T}(\texttt{n},\texttt{k})$	$= \texttt{T}(\texttt{n}-1,\texttt{k}) + \texttt{T}(\texttt{n}-1,\texttt{k}-1) + 1$

Relação de recorrência!

Relação de recorrência

$$T(n,k) = \begin{cases} 0, & n < k, \\ 0, & n = k \text{ ou } k = 0, \\ T(n-1,k) + T(n-1,k-1) + 1, & n,k > 0. \end{cases}$$

Quanto vale $T(n,k)$?

$$\binom{n}{k} = \begin{cases} 0, & \text{quando } n < k, \\ 1, & \text{quando } n = k \text{ ou } k = 0, \\ \binom{n-1}{k} + \binom{n-1}{k-1}, & \text{quando } n,k > 0. \end{cases}$$

Número $T(n, k)$ de adições

T	0	1	2	3	4	5	6	7	8	...	k
0	0	0	0	0	0	0	0	0	0	...	
1	0	0	0	0	0	0	0	0	0	...	
2	0	1	0	0	0	0	0	0	0	...	
3	0	2	2	0	0	0	0	0	0	...	
4	0	3	5	3	0	0	0	0	0	...	
5	0	4	9	9	4	0	0	0	0	...	
6	0	5	14	19	14	5	0	0	0	...	
7	0	6	20	34	34	20	6	0	0	...	
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋱	
n											

Binomial

	0	1	2	3	4	5	6	7	8	...	k
0	1	0	0	0	0	0	0	0	0	...	
1	1	1	0	0	0	0	0	0	0	...	
2	1	2	1	0	0	0	0	0	0	...	
3	1	3	3	1	0	0	0	0	0	...	
4	1	4	6	4	1	0	0	0	0	...	
5	1	5	10	10	5	1	0	0	0	...	
6	1	6	15	20	15	6	1	0	0	...	
7	1	7	21	35	35	21	7	1	0	...	
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋱	
n											

Número de adições

O número `T(n,k)` de adições feitas pela chamada `binomialR1(n,k)` é

$$\binom{n}{k} - 1 \,.$$

O consumo de tempo da função é proporcional ao número de iterações, logo é "*proporcional*" a $\binom{n}{k}$.

Quando o valor de `k` é aproximadamente `n`/2

$$\binom{n}{k} \geq 2^{\frac{n}{2}}$$

e o consumo de tempo é dito "*exponencial*".

Conclusões

Devemos **evitar** **resolver**
o mesmo subproblema várias vezes.

O número de chamadas recursivas feitas por `binomialR1(n,k)` é

$$2 \times \binom{\texttt{n}}{\texttt{k}} - 2\,.$$

Binomial mais eficiente ainda . . .

Supondo $\mathtt{n} \geq \mathtt{k} \geq 1$ temos que

$$\begin{aligned}
\binom{\mathtt{n}}{\mathtt{k}} &= \frac{\mathtt{n}!}{(\mathtt{n}-\mathtt{k})!\,\mathtt{k}!} \\
&= \frac{(\mathtt{n}-1)!}{(\mathtt{n}-\mathtt{k})!(\mathtt{k}-1)!} \times \frac{\mathtt{n}}{\mathtt{k}} \\
&= \frac{(\mathtt{n}-1)!}{((\mathtt{n}-1)-(\mathtt{k}-1))!(\mathtt{k}-1)!} \times \frac{\mathtt{n}}{\mathtt{k}} \\
&= \binom{\mathtt{n}-1}{\mathtt{k}-1} \times \frac{\mathtt{n}}{\mathtt{k}}\,.
\end{aligned}$$

Binomial mais eficiente ainda . . .

Logo, supondo $\texttt{n} \geq \texttt{k} \geq 1$, podemos escrever

$$\binom{\texttt{n}}{\texttt{k}} = \begin{cases} \texttt{n}, & \text{quando } \texttt{k} = 1, \\ \binom{\texttt{n}-1}{\texttt{k}-1} \times \frac{\texttt{n}}{\texttt{k}}, & \text{quando } \texttt{k} > 1. \end{cases}$$

```
 long binomialR2(int n, int k) {
1   if (k == 1) return n;
2   return binomialR2(n-1, k-1) * n / k;
 }
```

A função `binomialR2` faz
recursão de cauda (*tail recursion*).

binomialR2(20,10)

```
binomialR2(20,10)
  binomialR2(19,9)
    binomialR2(18,8)
      binomialR2(17,7)
        binomialR2(16,6)
          binomialR2(15,5)
            binomialR2(14,4)
              binomialR2(13,3)
                binomialR2(12,2)
                  binomialR2(11,1)
binom(20,10)=184756.
```

E agora, qual é mais eficiente?

```
meu_prompt> time ./binomialI 30 2
binom(30,2)=435
real                    0m0.003s
user                    0m0.001s
sys                     0m0.001s

meu_prompt> time ./binomialR2 30 2
binom(30,2)=435
real                    0m0.003s
user                    0m0.001s
sys                     0m0.001s
```

E agora, qual é mais eficiente?

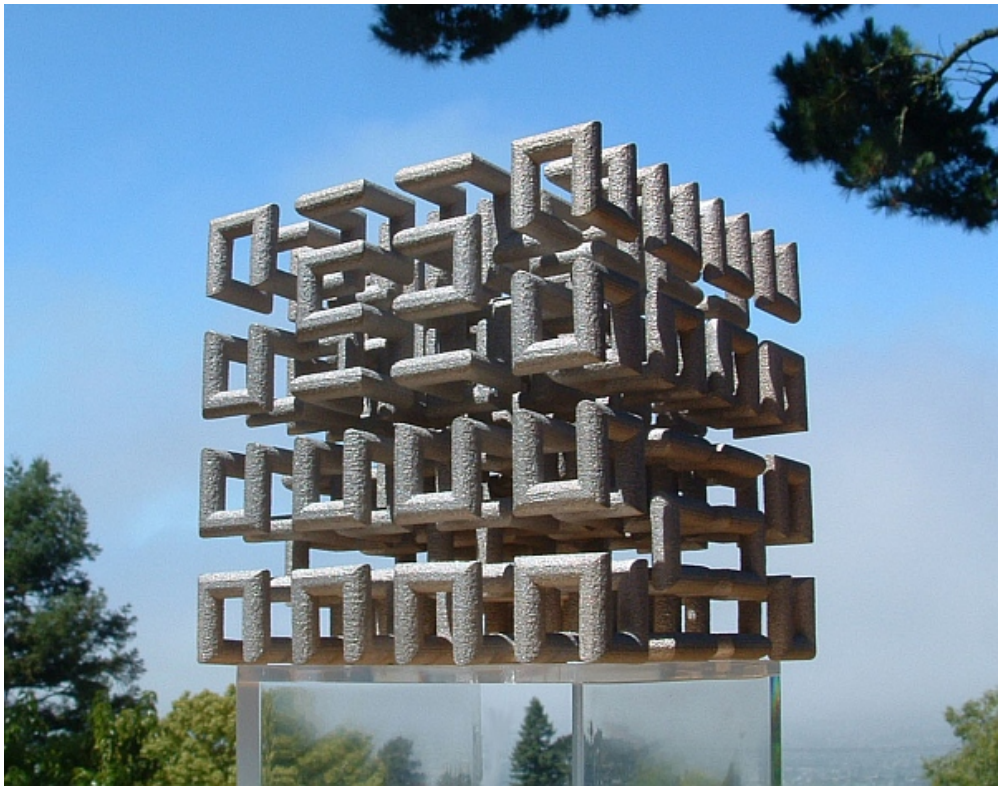
```
meu_prompt> time ./binomialI 40 30
binom(40,30)=847660528
real                                0m0.003s
user                                0m0.001s
sys                                 0m0.001s

meu_prompt> time ./binomialR2 40 30
binom(40,30)=847660528
real                                0m0.003s
user                                0m0.001s
sys                                 0m0.001s
```

Conclusão

O número de chamadas recursivas feitas por `binomialR2(n,k)` é $k - 1$.

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.

Curvas de Hilbert

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

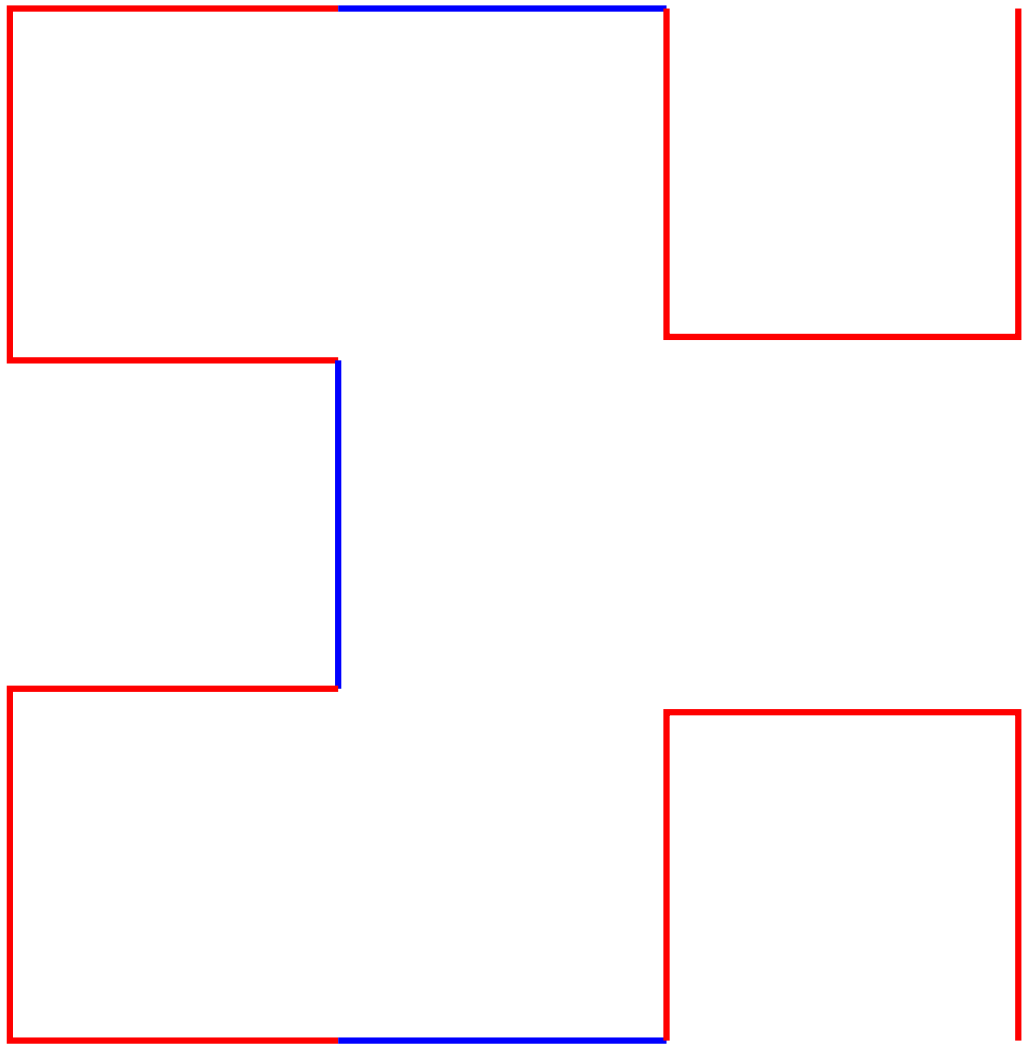
O objetivo é descobrir o esquema de recursão para construir tais curvas.

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

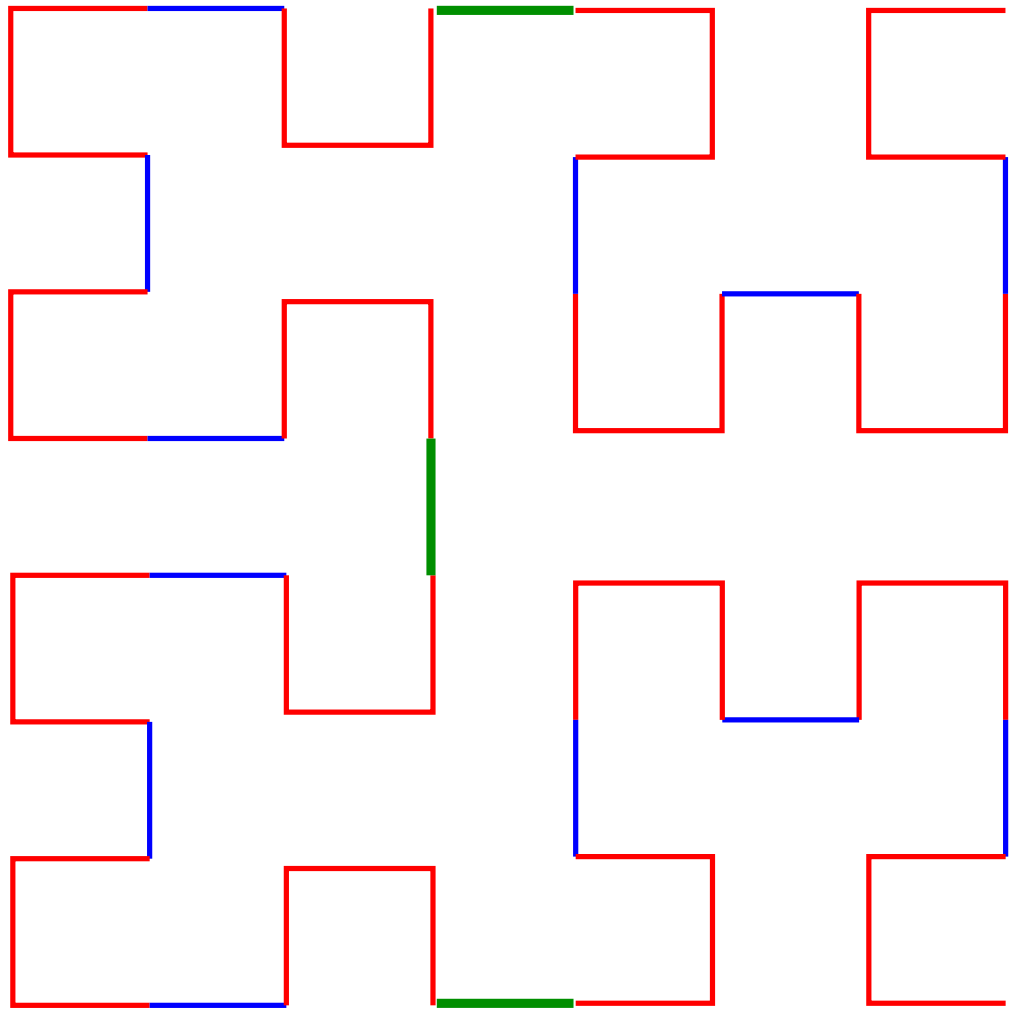
Cada $\mathtt{H}_\mathtt{i}$ denomina a **curva de Hilbert** de ordem $\mathtt{i}$, em homenagem a seu inventor, o matemático *David Hilbert*.

H_1

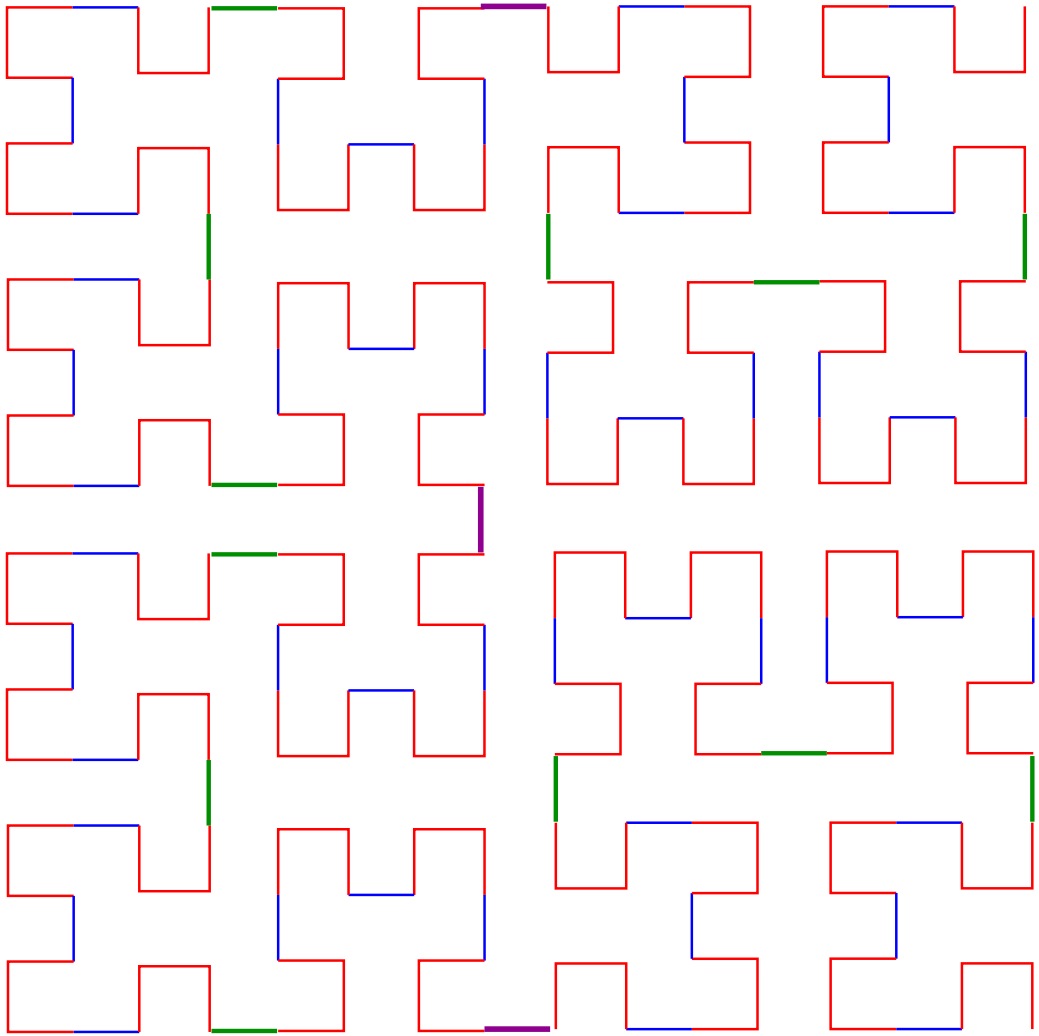
H_2



H_3



H_4



Qual é o padrão?

Vamos ver uma animação?

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.

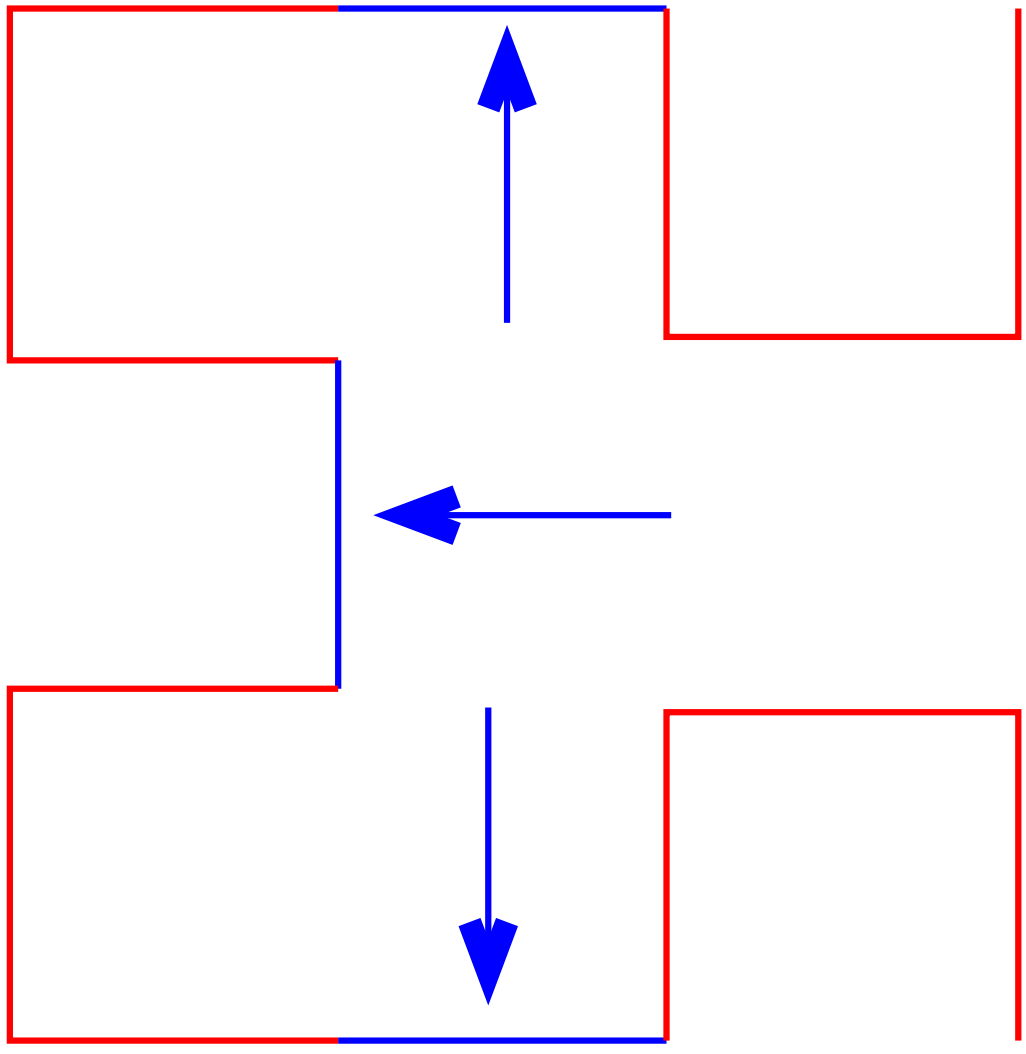
Padrão

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.

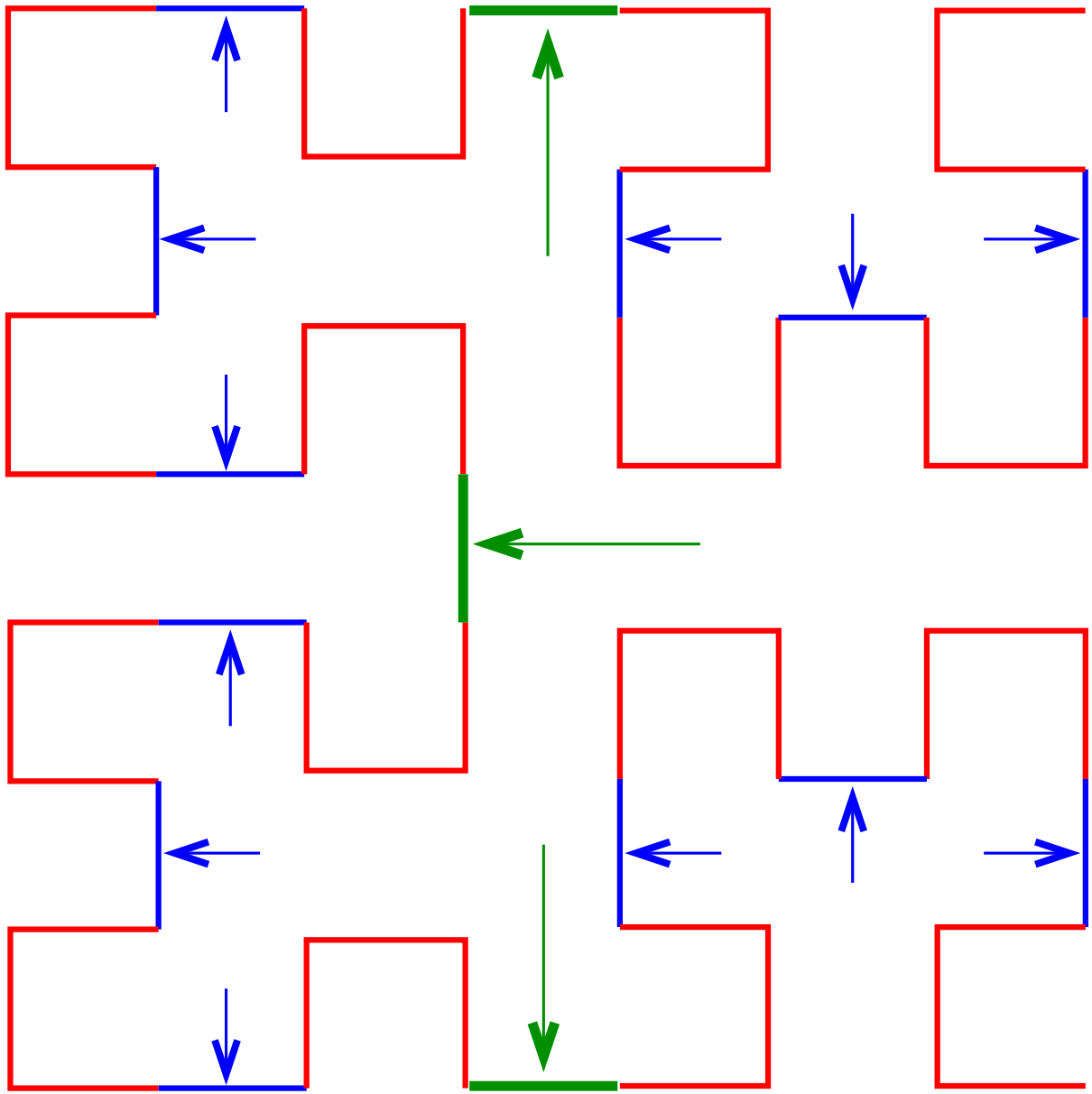
Por exemplo:

- 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

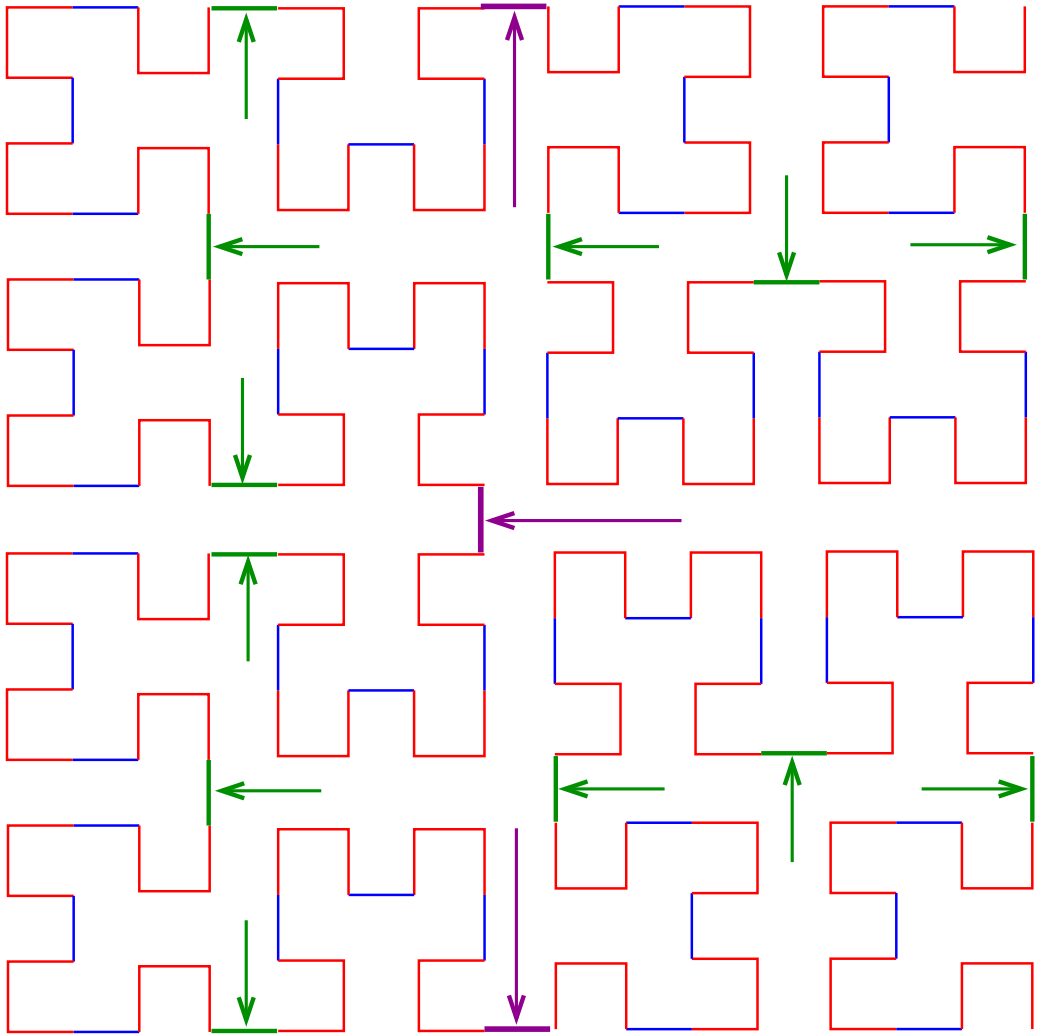
H_2



H_3



H_4



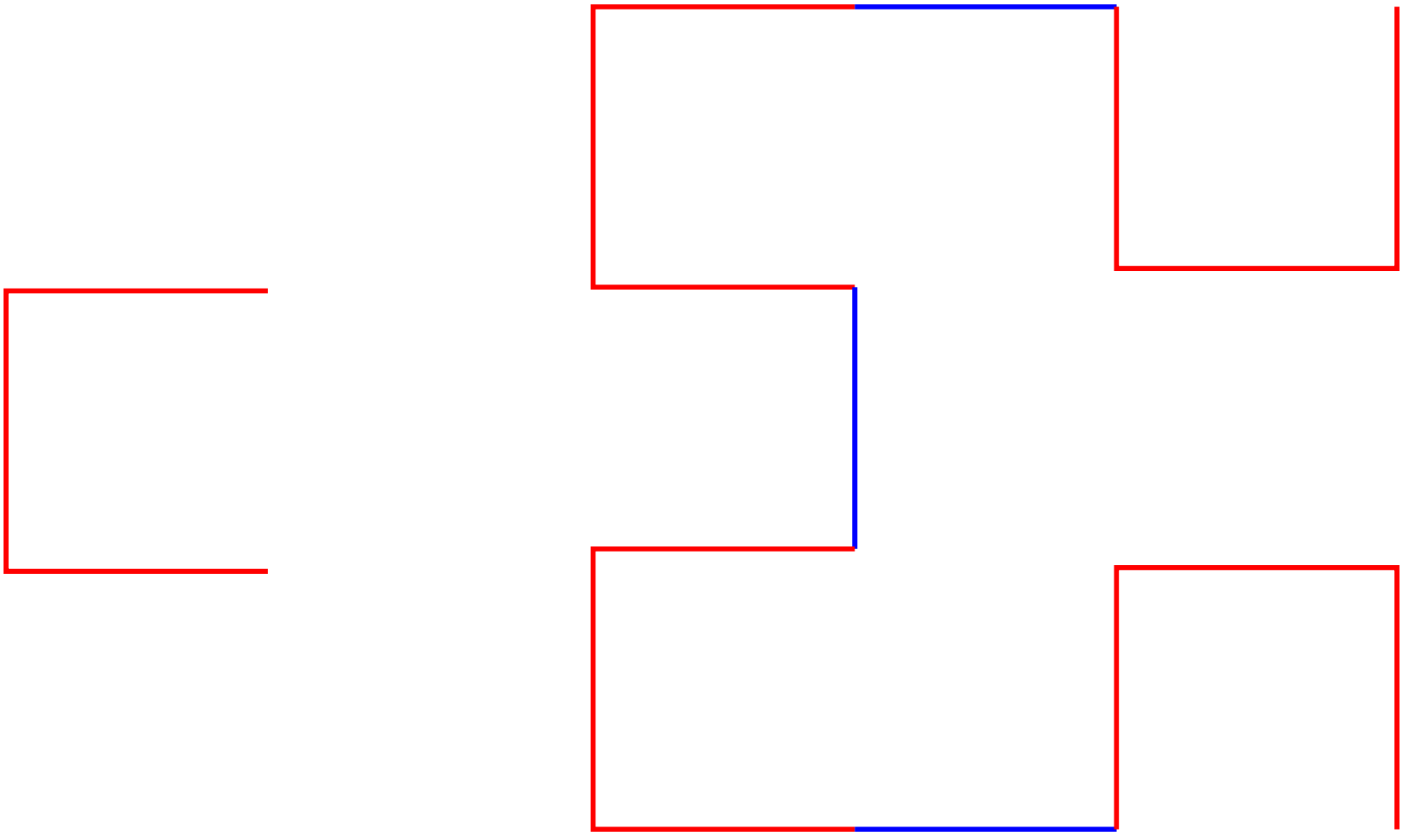
Partes da curva

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

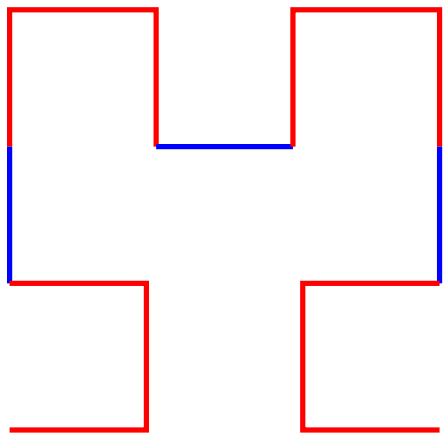
- `A` será o padrão que tem a "abertura" para direita;
- `B` será o padrão que tem a "abertura" para baixo;
- `C` será o padrão que tem a "abertura" para esquerda; e
- `D` será o padrão que tem a "abertura" para cima.

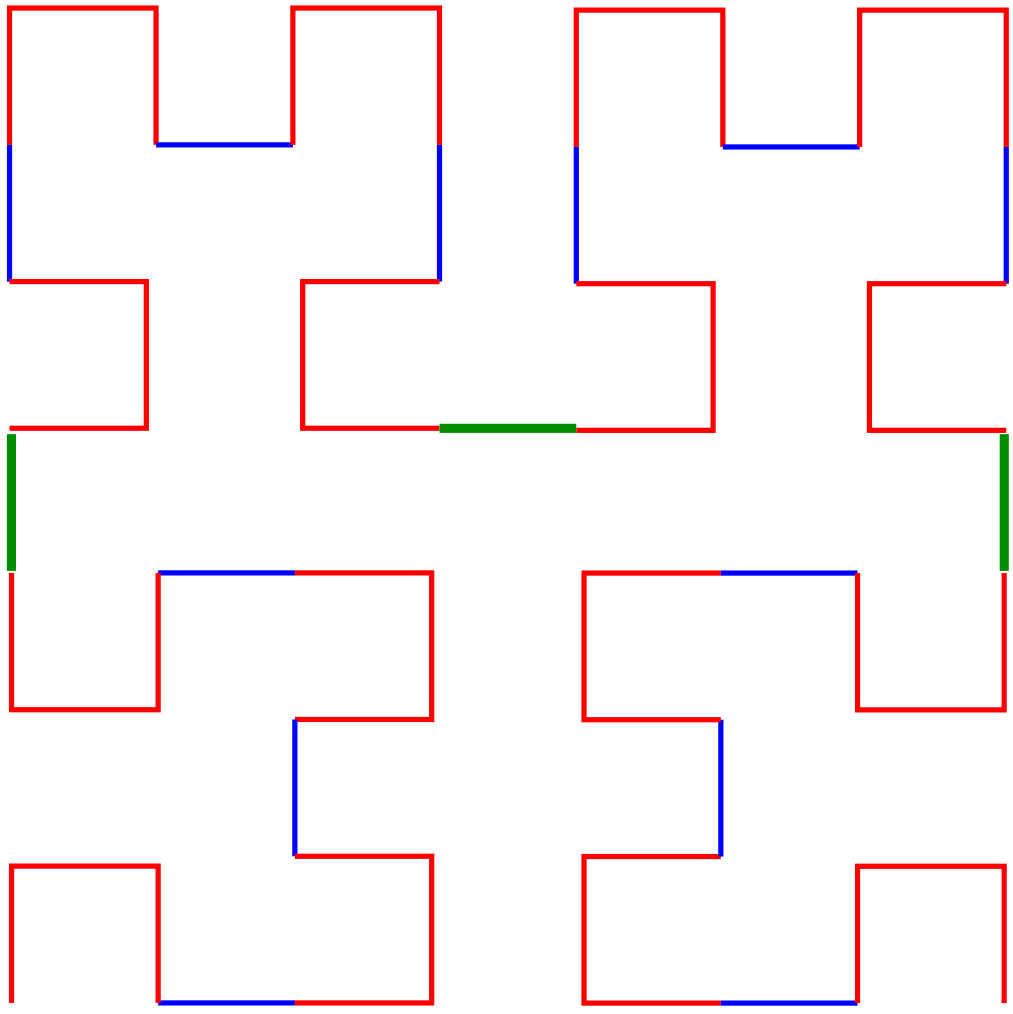
Representaremos a chamada da função que desenha as interconexões por meio da setas $\uparrow, \downarrow, \leftarrow, \rightarrow$.

A_1 e A_2

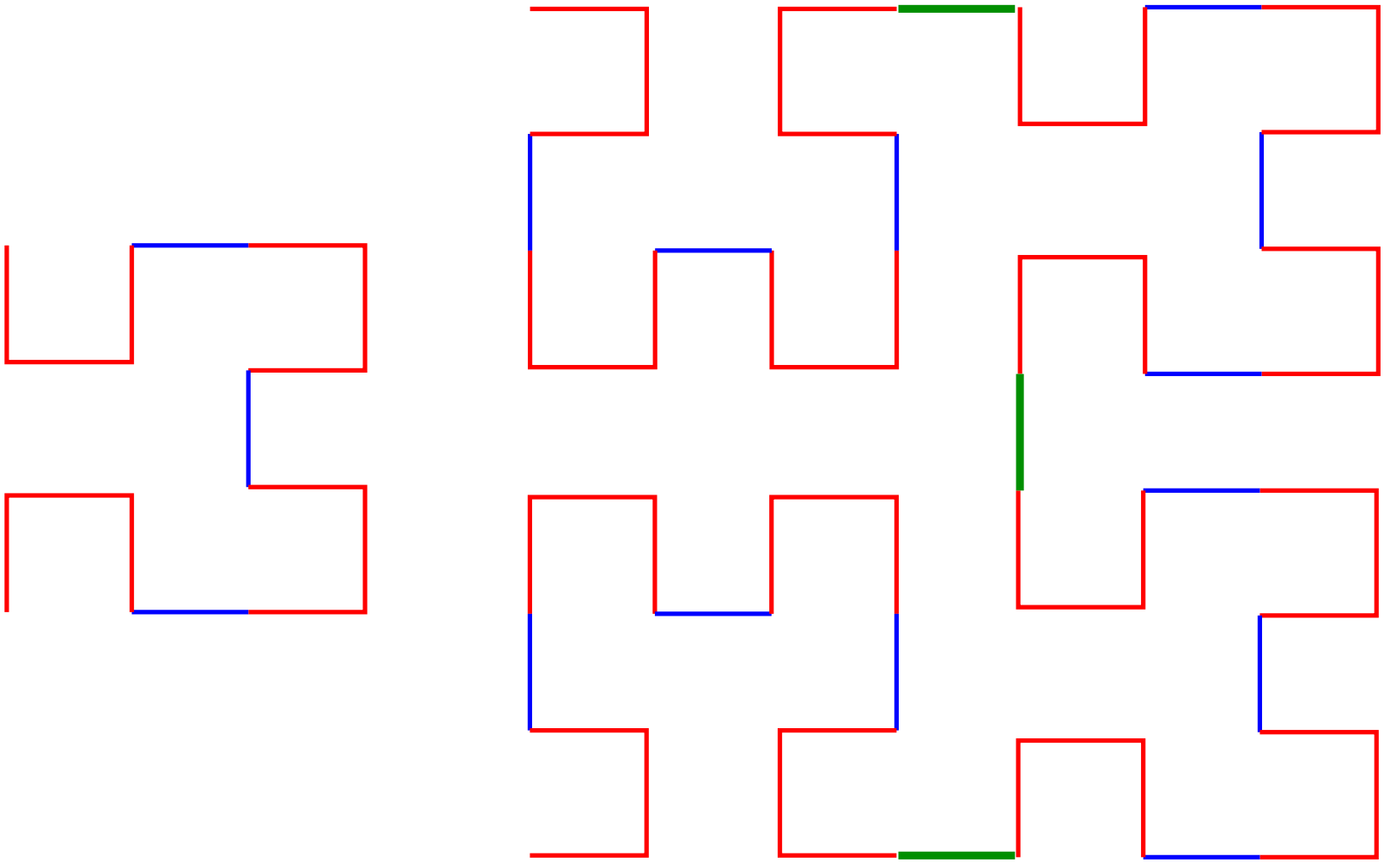


B_2 e B_3

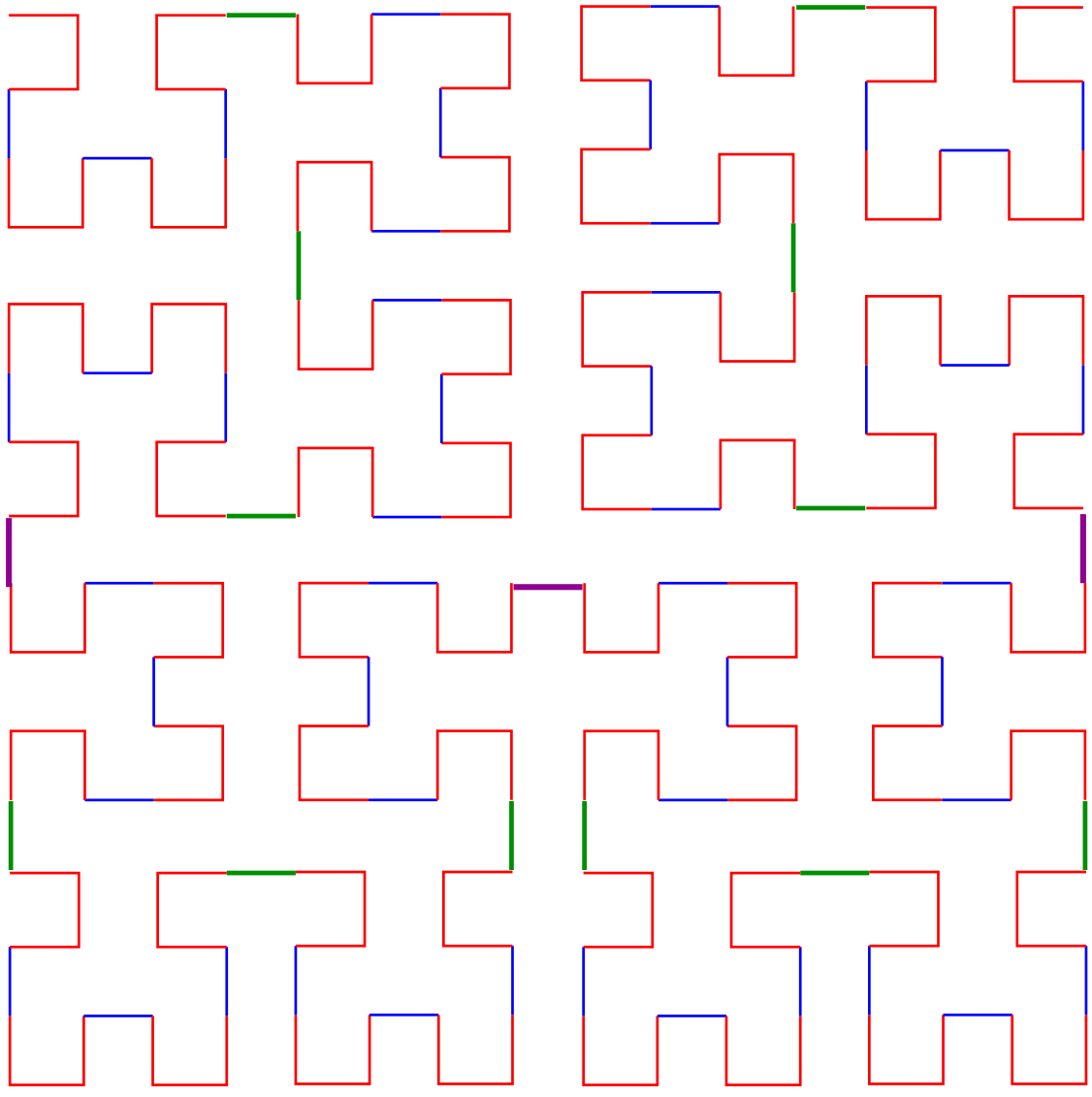




C_2 e C_3



D_4



Esquema recursivo

Assim, surge o seguinte esquema recursivo:

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

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;
   }
   desenheLinha(*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);
   }
}
```