Melhores momentos

AULA 1

Recursão

A resolução recursiva de um problema
tem tipicamente a seguinte estrutura:

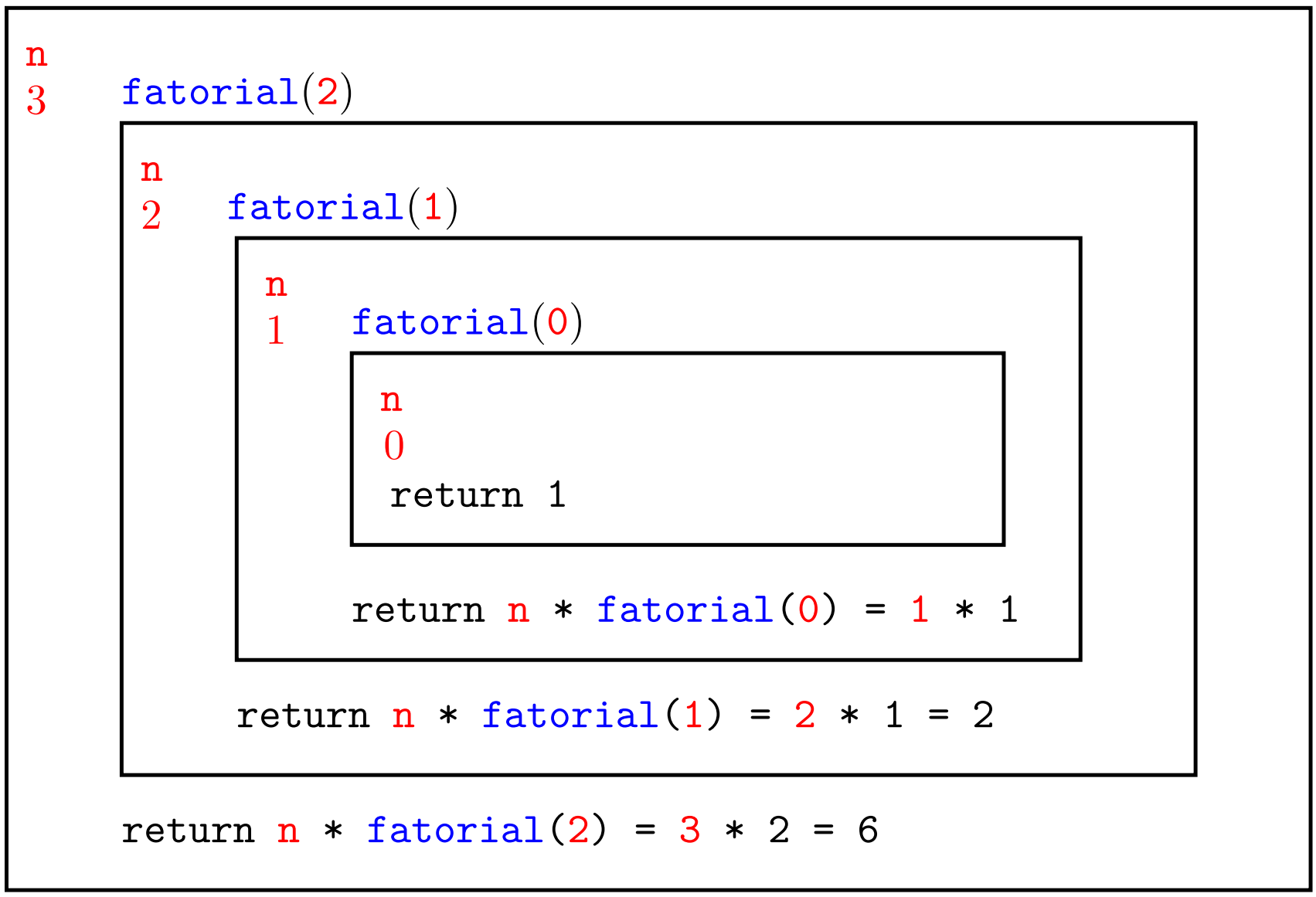
```
se a instância em questão é "pequena"
   resolva-a diretamente (use força bruta
   se necessário);
senão
   reduza-a a uma instância "menor" do
   mesmo problema,
   aplique o método à instância menor e
   volte à instância original.
```

Fatorial recursivo

$$\mathtt{n}! = \begin{cases} 1, & \text{quando } \mathtt{n} = 0, \\ \mathtt{n} \times (\mathtt{n} - 1)!, & \text{quando } \mathtt{n} > 0. \end{cases}$$

```
long fatorial(long n) {
   if (n == 0) return 1;
   return n * fatorial(n-1);
}
```

Diagramas de execução



Fatorial iterativo

```
long fatorial(long n) {
   int i, ifat;

   ifat = 1;
   for (i = 1; /*1*/ i <= n; i++)
       ifat *= i;

   return ifat;
}
```

Em `/*1*/` vale que `ifat == (i-1)!`

Exercícios

Escreva uma função recursiva que recebe como parâmetros:

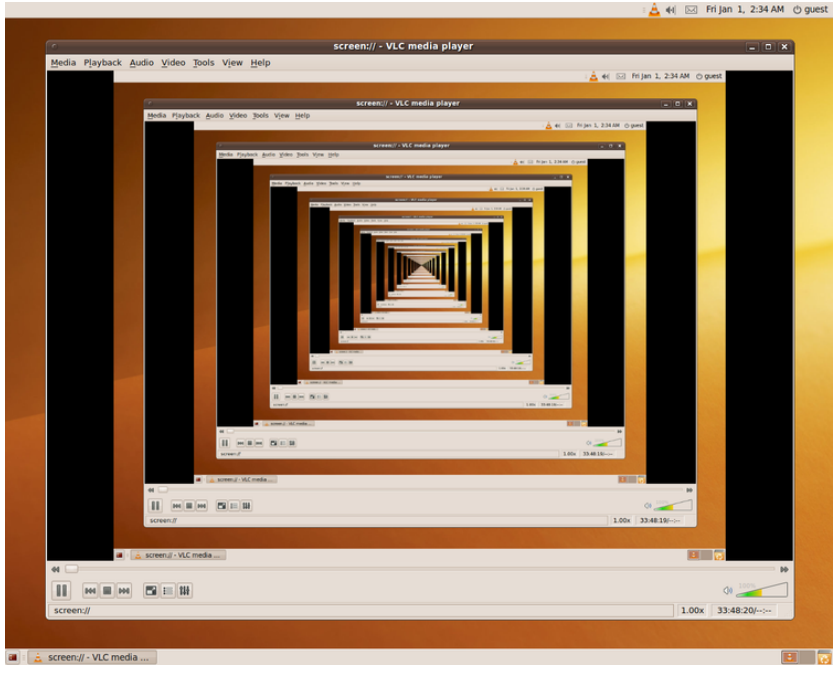
- um inteiro $n \geq 0$ e
- um vetor v com n números inteiros

e devolve o valor do maior inteiro no vetor $v[0\,..\,n-1]$.

Curva de Koch

AULA 2

Mais recursão



Fonte: http://commons.wikimedia.org/

PF 2.1, 2.2, 2.3 S 5.1

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

Problema do máximo

PF 2.2 e 2.3

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

Problema do máximo

Problema: encontrar o valor de um elemento máximo de um vetor `v[0 . . n-1]`.

Entra:

	0			4						n-1
v	10	44	10	35	99	40	20	65	55	50	38

Sai: `máximo == 99`

Máximo recursivo

```
 int maximoR(int n, int v[]) {
1   if (n == 1)
2       return v[0];
```

Máximo recursivo

```
 int maximoR(int n, int v[]) {
1   if (n == 1)
2       return v[0];
3   else {
4       int x;
5       x = maximoR(n-1, v);
```

Máximo recursivo

```
 int maximoR(int n, int v[]) {
1   if (n == 1)
2       return v[0];
3   else {
4       int x;
5       x = maximoR(n-1, v);
6       if (x > v[n-1])
7           return x;
8       else
9           return v[n-1];
    }
 }
```

. . . mais compactamente . . .

```
 int maximoR(int n, int v[]) {
0   int x;
1   if (n == 1) return v[0];
2   x = maximoR(n-1, v);
3   if (x > v[n-1]) return x;
4   return v[n-1];
 }
```

Outro máximo recursivo

```
 int maximo(int n, int v[]) {
1   return maxR(0, n, v);
 }
```

Outro máximo recursivo

```
 int maximo(int n, int v[]) {
1   return maxR(0, n, v);
 }

 int maxR(int i, int n, int v[]) {
1   if (i == n-1) return v[i];
```

Outro máximo recursivo

```
 int maximo(int n, int v[]) {
1   return maxR(0, n, v);
 }

 int maxR(int i, int n, int v[]) {
1   if (i == n-1) return v[i];
3   else {
4       int x;
5       x = maxR(i+1, n, v);
```

Outro máximo recursivo

```
 int maximo(int n, int v[]) {
1   return maxR(0, n, v);
 }

 int maxR(int i, int n, int v[]) {
1   if (i == n-1) return v[i];
3   else {
4       int x;
5       x = maxR(i+1, n, v);
6       if (x > v[i]) return x;
7       else return v[i];
    }
 }
```

. . . alternativamente . . .

```
 int maximo(int n, int v[]) {
1   return maxR(0, n, v);
 }

 int maxR(int i, int n, int v[]) {
0   int x;

1   if (i == n-1) return v[i];
2   x = maxR(i+1, n, v);
3   if (x > v[i]) return x;
4   return v[i];
 }
```

. . . alternativamente . . .

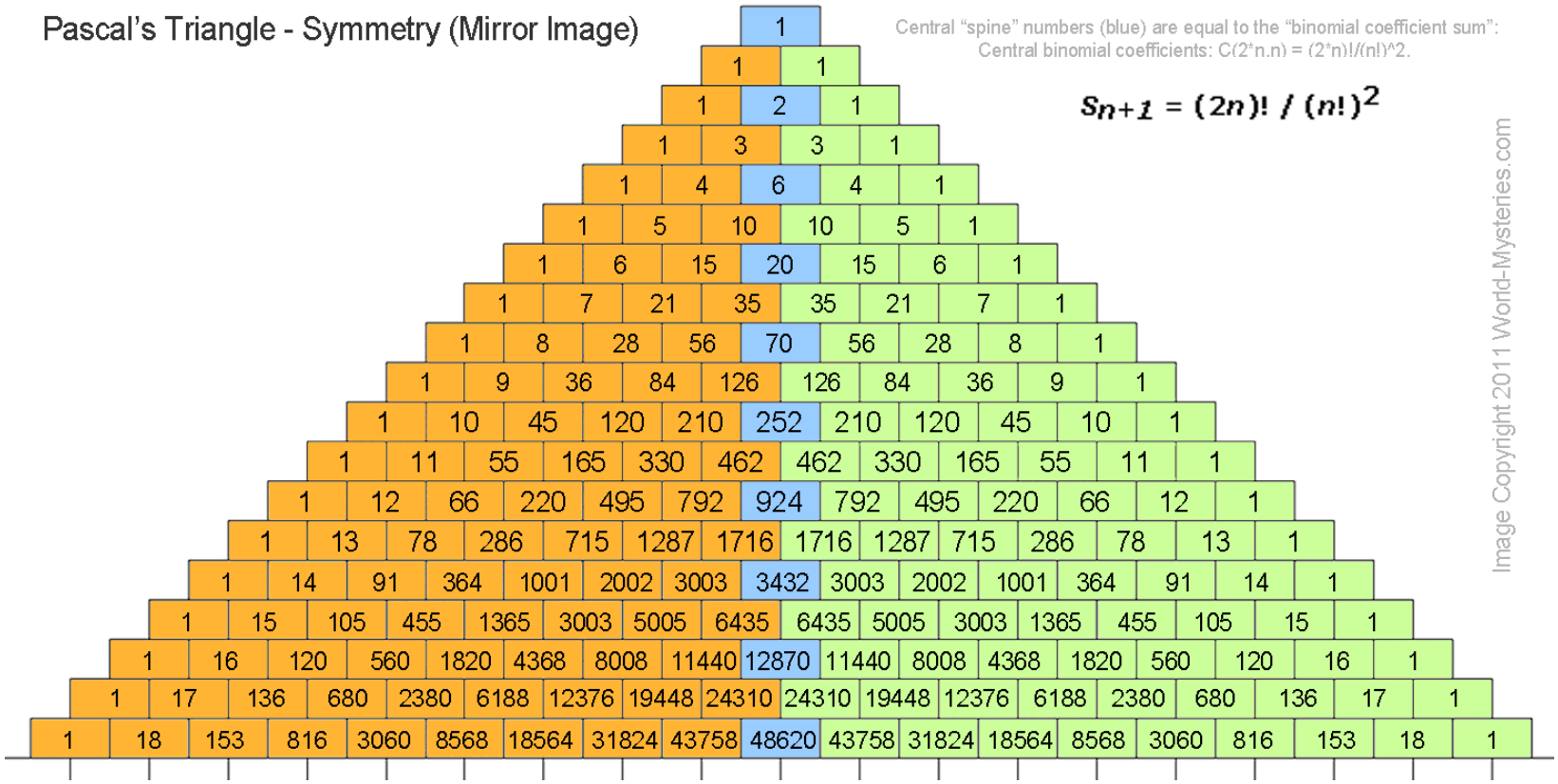
```
 int maximo(int n, int v[]) {
1   return maxR(0, n, v);
 }

 int maxR(int i, int n, int v[]) {
0   int x;

1   if (i == n-1) return v[i];
2   x = maxR(i+1, n, v);
3   if (x > v[i]) return x;
4   return v[i];
 }
```

Beleza?

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

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

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 recursivo

Binomial recursivo

```
 long binomialR0(int n, int k) {
1   if (n == 0 && k >  0) return 0;
2   if (n >= 0 && k == 0) return 1;
```

Binomial recursivo

```
 long binomialRO(int n, int k) {
1   if (n == 0 && k >  0) return 0;
2   if (n >= 0 && k == 0) return 1;
3   return binomialRO(n-1, k) +
4          binomialRO(n-1, k-1);
 }
```

binomialR0(3,2)

```
binomialR0(3,2)
  binomialR0(2,2)
    binomialR0(1,2)
      binomialR0(0,2)
      binomialR0(0,1)
    binomialR0(1,1)
      binomialR0(0,1)
      binomialR0(0,0)
  binomialR0(2,1)
    binomialR0(1,1)
      binomialR0(0,1)
      binomialR0(0,0)
    binomialR0(1,0)
binom(3,2)=3.
```

Binomial iterativo

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;
```

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];
}
```

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 ./binomialR0 30 2
binom(30,2)=435
real                    0m0.003s
user                    0m0.001s
sys                     0m0.001s
```

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 ./binomialR0 30 20
binom(30,20)=30045015
real                                0m5.426s
user                                0m5.379s
sys                                 0m0.010s
```

Resolve subproblemas muitas vezes

```
binomialR0(3,2)
  binomialR0(2,2)
    binomialR0(1,2)
      binomialR0(0,2)
      binomialR0(0,1)
    binomialR0(1,1)
      binomialR0(0,1)
      binomialR0(0,0)
  binomialR0(2,1)
    binomialR0(1,1)
      binomialR0(0,1)
      binomialR0(0,0)
    binomialR0(1,0)
binom(3,2)=3.
```

Resolve subproblemas muitas vezes

```
binomialR0(5,4)
  binomialR0(4,4)
    binomialR0(3,4)
      binomialR0(2,4)
        binomialR0(1,4)
          binomialR0(0,4)
          binomialR0(0,3)
        binomialR0(1,3)
          binomialR0(0,3)
          binomialR0(0,2)
      binomialR0(2,3)
        binomialR0(1,3)
          binomialR0(0,3)
          binomialR0(0,2)
        binomialR0(1,2)
          binomialR0(0,2)
          binomialR0(0,1)
    binomialR0(3,3)
      binomialR0(2,3)
        binomialR0(1,3)
          binomialR0(0,3)
        binomialR0(0,2)
      binomialR0(1,2)
        binomialR0(0,2)
        binomialR0(0,1)
    binomialR0(2,2)
      binomialR0(1,2)
        binomialR0(0,2)
        binomialR0(0,1)
      binomialR0(1,1)
        binomialR0(0,1)
        binomialR0(0,0)
  binomialR0(4,3)
    binomialR0(3,3)
      binomialR0(2,3)
        binomialR0(1,3)
          binomialR0(0,3)
          binomialR0(0,2)
        binomialR0(1,2)
          binomialR0(0,2)
          binomialR0(0,1)
      binomialR0(2,2)
        binomialR0(1,2)
          binomialR0(0,2)
          binomialR0(0,1)
        binomialR0(1,1)
          binomialR0(0,1)
          binomialR0(0,0)
    binomialR0(3,2)
      binomialR0(2,2)
        binomialR0(1,2)
          binomialR0(0,2)
          binomialR0(0,1)
        binomialR0(1,1)
          binomialR0(0,1)
          binomialR0(0,0)
      binomialR0(2,1)
        binomialR0(1,1)
          binomialR0(0,1)
          binomialR0(0,0)
        binomialR0(1,0)
binom(5,4)=5.
```

Mais eficiente . . .

Versão anterior da regra de Pascal:

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

Outro jeito de definir a regra de Pascal:

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

Mais eficiente . . .

```
 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);
 }
```

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

Resolve subproblemas muitas vezes?

```
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.
```

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)
```

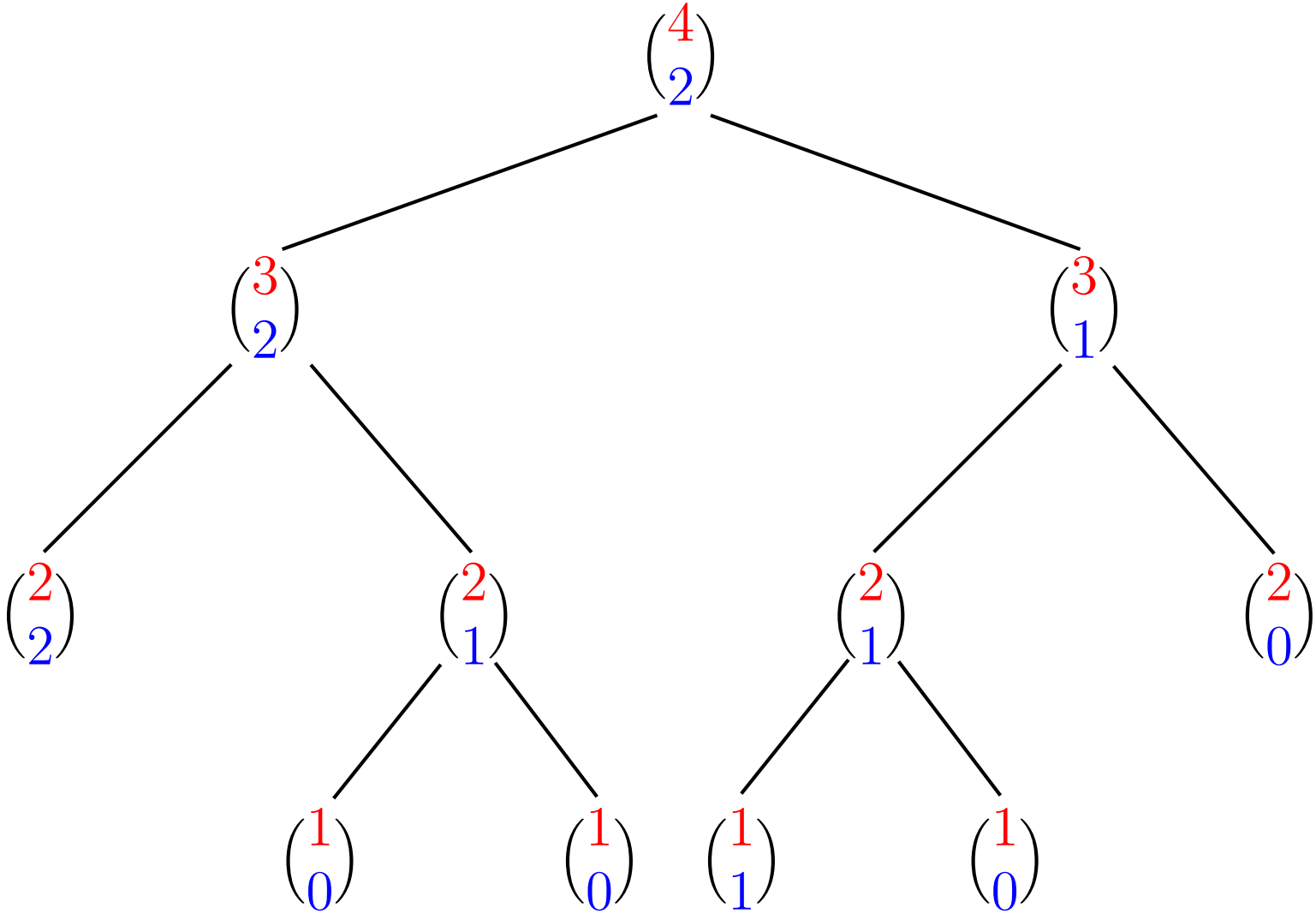
```
            binomialR1(1,0)
        binomialR1(3,1)
          binomialR1(2,1)
            binomialR1(1,1)
            binomialR1(1,0)
          binomialR1(2,0)
  binomialR1(6,3)
    binomialR1(5,3)
      binomialR1(4,3)
        binomialR1(3,3)
        binomialR1(3,2)
          binomialR1(2,2)
  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

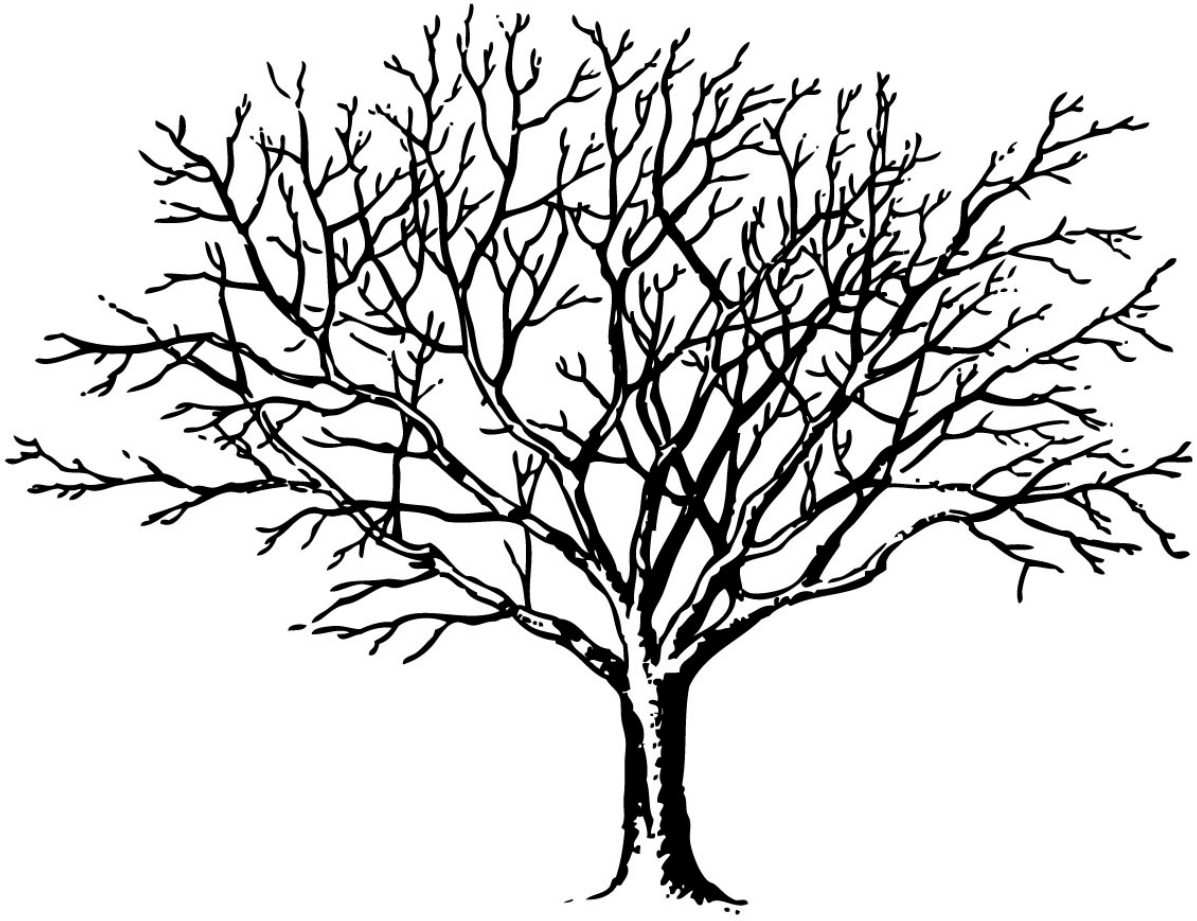
Sim!

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);
 }
```

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.

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

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

Quanto vale $\mathtt{T}(\mathtt{n},\mathtt{k})$?

Relação de recorrência

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

Quanto vale $\mathtt{T}(\mathtt{n},\mathtt{k})$?

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

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

$$\binom{\mathtt{n}}{\mathtt{k}} = \frac{\mathtt{n}!}{(\mathtt{n}-\mathtt{k})!\,\mathtt{k}!}$$

Binomial mais eficiente ainda . . .

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

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

Binomial mais eficiente ainda . . .

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

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

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 $\mathtt{n} \geq \mathtt{k} \geq 1$, podemos escrever

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

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

Binomial mais eficiente ainda . . .

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

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

```
 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?

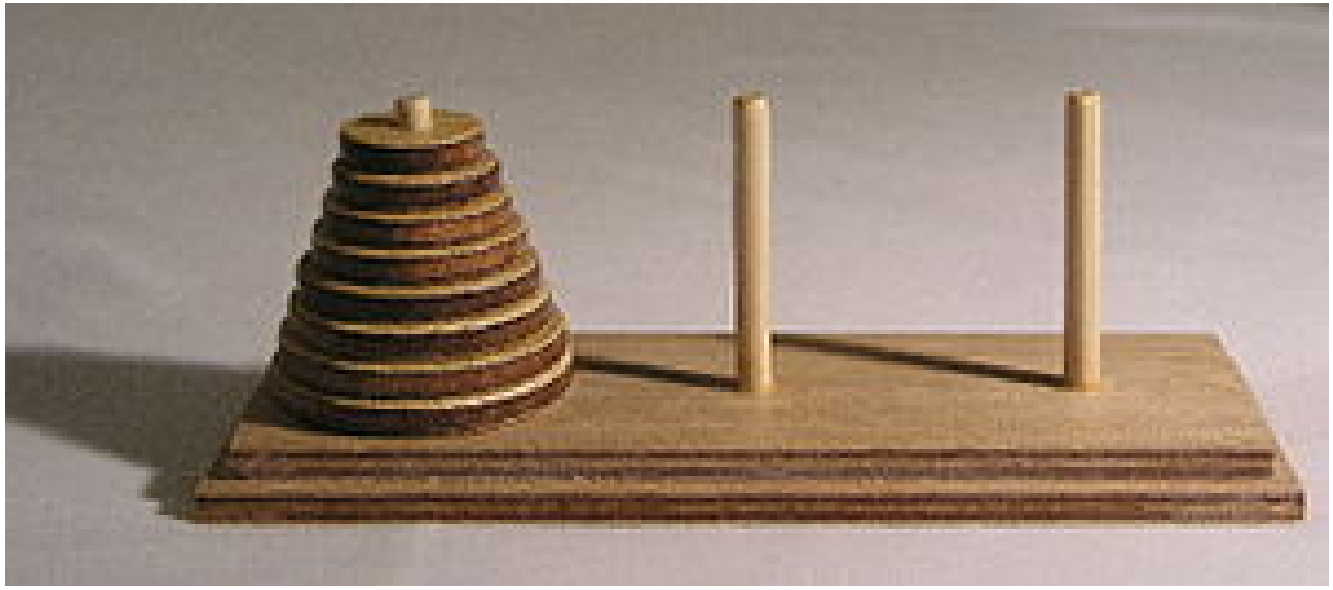
```
meu_prompt> time ./binomialI 40 30
binom(40,30)=847660528
real                                0m0.003s
user                                0m0.001s
sys                                 0m0.001s

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

Conclusão

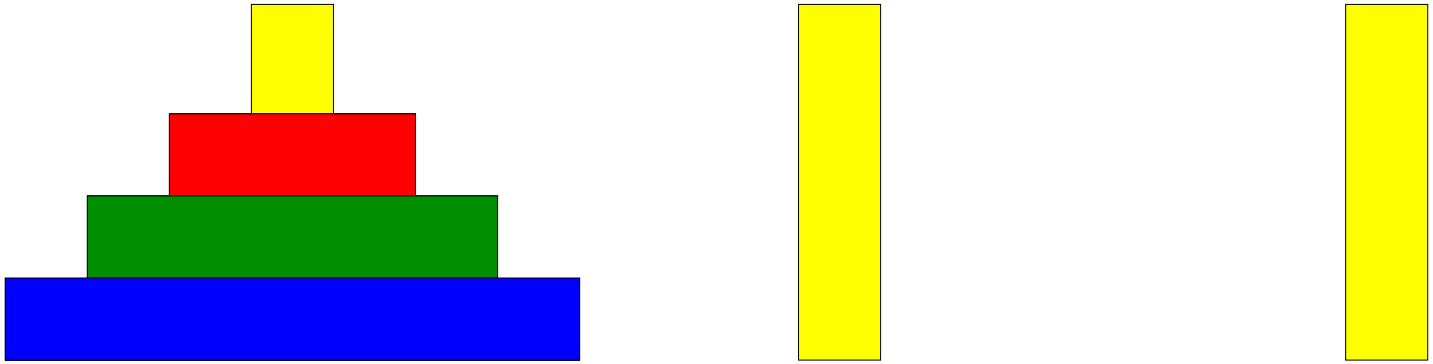
O número de chamadas recursivas feitas por `binomialR2(n,k)` é $k - 1$.

Torres de Hanoi: epílogo



Fonte: http://en.wikipedia.org/

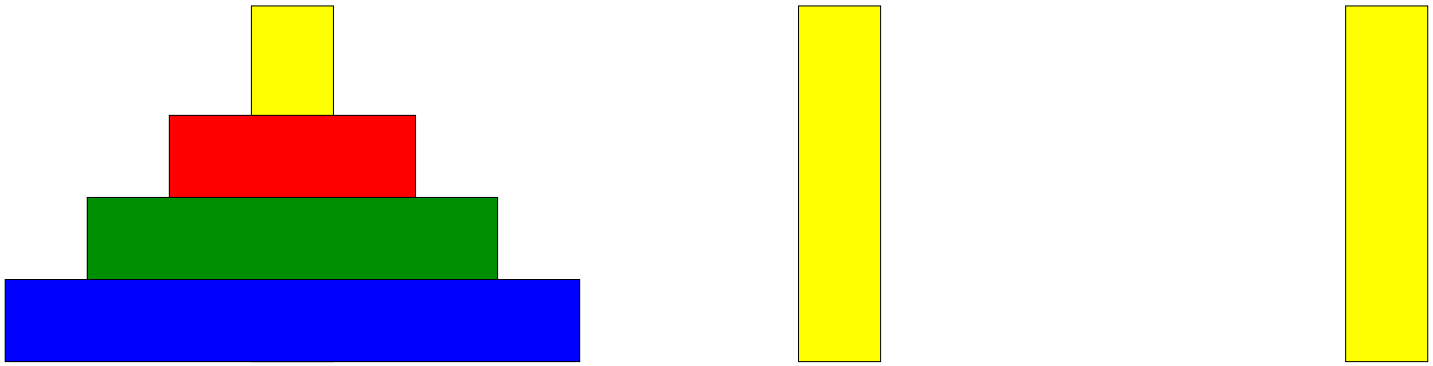
Torres de Hanoi



Desejamos transferir `n` discos do pino `A` para o pino `C` usando o pino `B` como auxiliar e repeitando as regras:

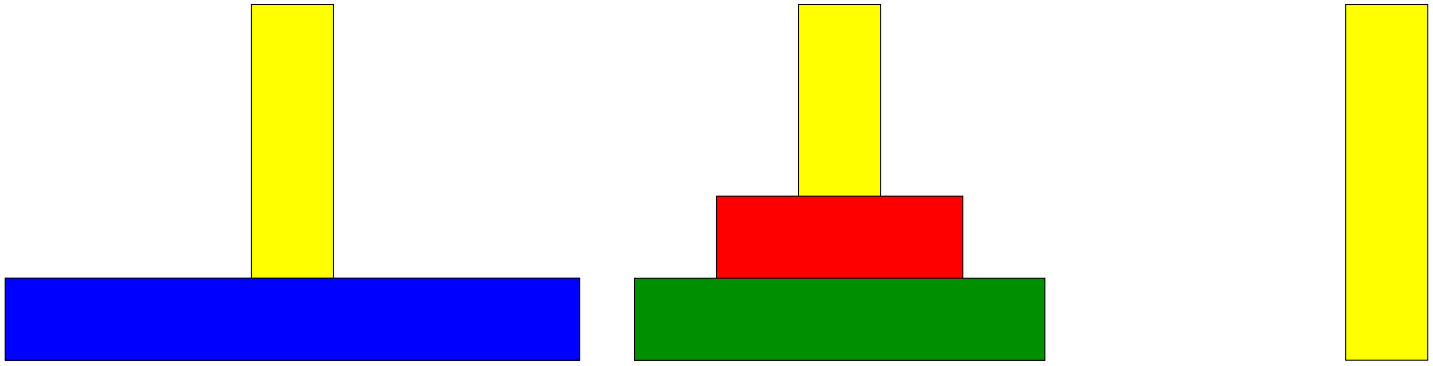
- podemos mover apenas um disco por vez;
- nunca um disco de diâmetro maior poderá ser colocado sobre um disco de diâmetro menor.

Algoritmo recursivo



Para resolver HANOI(n,A,B,C) basta:

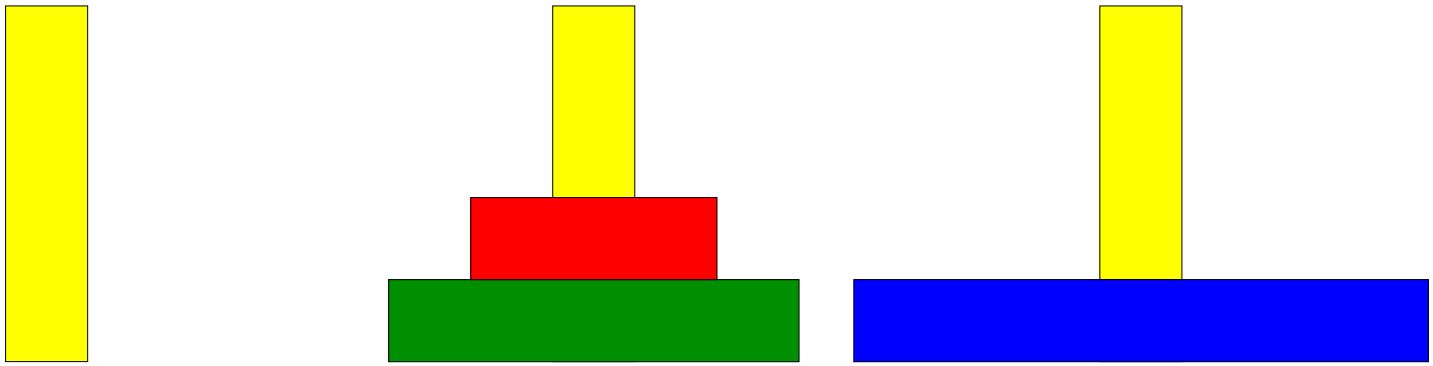
Algoritmo recursivo



Para resolver HANOI(n,A,B,C) basta:

1. resolver HANOI(n-1,A,C,B)

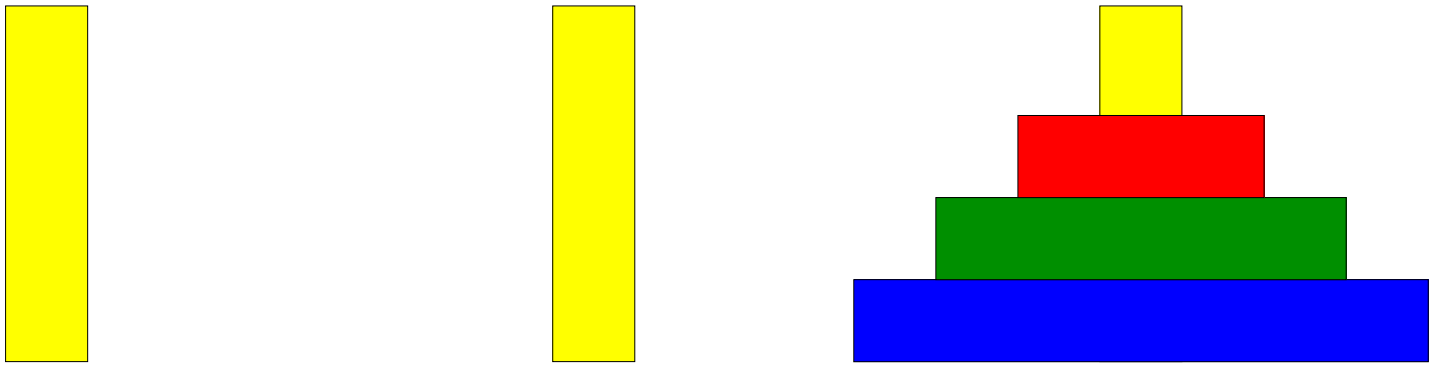
Algoritmo recursivo



Para resolver HANOI(n,A,B,C) basta:

1. resolver HANOI(n-1,A,C,B)
2. mover o disco n de A para C

Algoritmo recursivo



Para resolver HANOI(n,A,B,C) basta:

1. resolver HANOI(n-1,A,C,B)
2. mover o disco n de A para C
3. resolver HANOI(n-1,B,A,C)

Base: sabemos revolver HANOI(**0**,. . . ,. . . ,. . .)

Número de movimentos

Seja $\mathtt{T}(\mathtt{n})$ o número de movimentos feitos pelo algoritmo para resolver o problema das torres de HANOI com $\mathtt{n}$ disco.

Temos que

$$\mathtt{T}(0) = 0$$
$$\mathtt{T}(\mathtt{n}) = 2\,\mathtt{T}(\mathtt{n}-1) + 1 \quad \text{para } \mathtt{n} = 1, 2, 3 \ldots$$

Quanto vale $\mathtt{T}(\mathtt{n})$?

Recorrência

Temos que

$$
\begin{aligned}
\mathtt{T}(\mathtt{n}) &= 2\,\mathtt{T}(\mathtt{n}-1)+1 \\
&= 2\,(2\,\mathtt{T}(\mathtt{n}-2)+1)+1 \\
&= 2\,(2\,(2\,\mathtt{T}(\mathtt{n}-3)+1)+1)+1 \\
&= 2\,(2\,(2\,(2\,\mathtt{T}(\mathtt{n}-4)+1)+1)+1)+1 \\
&= \cdots \\
&= 2\,(2\,(2\,(2\,(\cdots(2\,\mathtt{T}(0)+1))+1)+1)+1)+1
\end{aligned}
$$

Recorrência

Logo,

$$\begin{aligned} \mathtt{T}(\mathtt{n}) &= 2^{\mathtt{n}-1} + \cdots + 2^3 + 2^2 + 2 + 1 \\ &= 2^{\mathtt{n}} - 1\,. \end{aligned}$$

n	0	1	2	3	4	5	6	7	8	9
T(n)	0	1	3	7	15	31	63	127	255	511

Conclusões

O número de movimentos feitos pela chamada `hanoi(n,...,...,...)` é

$$2^{\texttt{n}} - 1\,.$$

Notemos que a função `hanoi` faz o **número mínimo** de movimentos: **não é possível** resolver o quebra-cabeça com menos movimentos.

The Tower of Hanoi Story

Taken From W.W. Rouse Ball & H.S.M. Coxeter, Mathematical Recreations and Essays, 12th edition. Univ. of Toronto Press, 1974. The De Parville account of the origen from La Nature, Paris, 1884, part I, pp. 285-286.

> *In the great temple at Benares beneath the dome that marks the centre of the world, rests a brass plate in which are fixed three diamond needles, each a cubit high and as thick as the body of a bee. On one of these needles, at the creation, God placed sixty-four discs of pure gold, the largest disk resting on the brass plate, and the others getting smaller and smaller up to the top one. This is the tower of Bramah. Day and night unceasingly the priest transfer the discs from one diamond needle to another according to the fixed and immutable laws of Bramah, which require that the priest on duty must not move more than one disc at a time and that he must place this disc on a needle so that there is no smaller disc below it. When the sixty-four discs shall have been thus transferred from the needle which at creation God placed them, to one of the other needles, tower, temple, and Brahmins alike will crumble into dust and with a thunderclap the world will vanish . . .*

http://www.rci.rutgers.edu/~cfs/472_html/AI_SEARCH/Story_TOH.html

Enquanto isto . . . os monges . . .

$$\mathtt{T}(64) = 18.446.744.073.709.551.615 \approx 1{,}84 \times 10^{19}$$

Suponha que os monges façam
o movimento de 1 disco por segundo(!).

$$\begin{aligned} 18 \times 10^{19}\,\text{seg} &\approx 3{,}07 \times 10^{17}\,\text{min} \\ &\approx 5{,}11 \times 10^{15}\,\text{horas} \\ &\approx 2{,}13 \times 10^{14}\,\text{dias} \\ &\approx 5{,}83 \times 10^{11}\,\text{anos.} \\ &= \mathbf{583}\,\text{bilhões de anos.} \end{aligned}$$

A idade da Terra é **4,54 bilhões de anos**.

The Tower of Hanoi Story

Taken From W.W. Rouse Ball & H.S.M. Coxeter, Mathematical Recreations and Essays, 12th edition. Univ. of Toronto Press, 1974. The De Parville account of the origen from La Nature, Paris, 1884, part I, pp. 285-286.

> *In the great temple at Benares beneath the dome that marks the centre of the world, rests a brass plate in which are fixed three diamond needles, each a cubit high and as thick as the body of a bee. On one of these needles, at the creation, God placed sixty-four discs of pure gold, the largest disk resting on the brass plate, and the others getting smaller and smaller up to the top one. This is the tower of Bramah. Day and night unceasingly the priest transfer the discs from one diamond needle to another according to the fixed and immutable laws of Bramah, which require that the priest on duty must not move more than one disc at a time and that he must place this disc on a needle so that there is no smaller disc below it. When the sixty-four discs shall have been thus transferred from the needle which at creation God placed them, to one of the other needles, tower, temple, and Brahmins alike will crumble into dust and with a thunderclap the world will vanish. The number of separate transfers of single discs which the Brahmins must make to effect the transfer of the tower is two raised to the sixty-fourth power minus 1 or 18,446,744,073,709,551,615 moves. Even if the priests move one disk every second, it would take more than 500 billion years to relocate the initial tower of 64 disks.*

http://www.rci.rutgers.edu/~cfs/472_html/AI_SEARCH/Story_TOH.html