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

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

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

Diagramas de execução

```
fatorial(3)
+------------------------------------------------------------+
| n                                                          |
| 3   fatorial(2)                                            |
|     +--------------------------------------------------+   |
|     | n                                                |   |
|     | 2   fatorial(1)                                  |   |
|     |     +------------------------------------+       |   |
|     |     | n                                  |       |   |
|     |     | 1   fatorial(1)                    |       |   |
|     |     |     +------------------------+     |       |   |
|     |     |     | n                      |     |       |   |
|     |     |     | 0                      |     |       |   |
|     |     |     |  return 1              |     |       |   |
|     |     |     +------------------------+     |       |   |
|     |     |                                    |       |   |
|     |     |     return n * fatorial(0) = 1 * 1 |       |   |
|     |     +------------------------------------+       |   |
|     |                                                  |   |
|     |     return n * fatorial(1) = 2 * 1 = 2           |   |
|     +--------------------------------------------------+   |
|                                                            |
|     return n * fatorial(2) = 3 * 2 = 6                     |
+------------------------------------------------------------+
```

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)!`

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

. . . alternativamente . . .

```
int maximoR(int n, int v[])
{
   int x;

   if (n == 1) return v[0];
   x = maximoR(n-1, v);
   if (x > v[n-1]) return x;
   return v[n-1];
}
```

Outro máximo recursivo

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

int maxR(int i, int n, int v[])
{
   if (i == n-1) return v[i];
   else {
           int x;
           x = maximoR(i+1, n, v);
           if (x > v[i]) return x;
           else return v[i];
   }
}
```

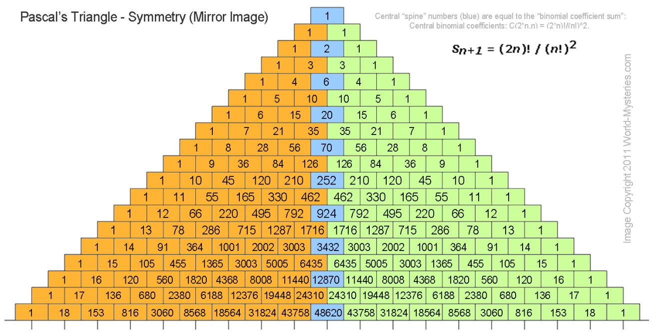
. . . alternativamente . . .

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

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

   if (i == n-1) return v[i];
   x = maximoR(i+1, n, v);
   if (x > v[i]) return x;
   return v[i];
}
```

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 recursivo

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

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

  for (j = 1; j <= k; j++) bin[0][k] = 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                    0m0.002s
user                    0m0.000s
sys                     0m0.000s

meu_prompt> time ./binomialR0 30 2
binom(30,2)=435
real                    0m0.002s
user                    0m0.000s
sys                     0m0.000s
```

Qual é mais eficiente?

```
meu_prompt> time ./binomialI 30 20
binom(30,20)=30045015
real                    0m0.002s
user                    0m0.000s
sys                     0m0.000s

meu_prompt> time ./binomialR0 30 20
binom(30,20)=30045015
real                    0m17.886s
user                    0m17.881s
sys                      0m0.000s
```

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 ...

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 . . .

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

E agora? Qual é mais eficiente?

```
meu_prompt> time ./binomialI 30 20
binom(30,20)=30045015
real                    0m0.002s
user                    0m0.000s
sys                     0m0.000s

meu_prompt> time ./binomialR1 30 20
binom(30,20)=30045015
real                    0m0.547s
user                    0m0.544s
sys                     0m0.000s
```

E agora? Qual é mais eficiente?

```
meu_prompt> time ./binomialI 40 30
binom(40,30)=847660528
real                    0m0.002s
user                    0m0.000s
sys                     0m0.000s

meu_prompt> time ./binomialR1 40 30
binom(40,30)=847660528
real                    0m14.001s
user                    0m13.997s
sys                      0m0.000s
```

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!

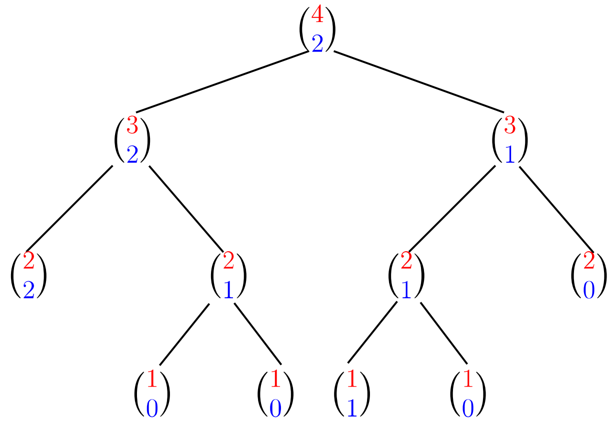
Resolve subproblemas muitas vezes?

```
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?

É o dobro do número de adições.

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

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

Número de adições

```
  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

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

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 $T(n,k)$?

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 e portanto 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

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

Binomial mais eficiente ainda ...

Supondo $n \geq k \geq 1$ temos que

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

Binomial mais eficiente ainda ...

Logo, supondo $n \geq k \geq 1$, podemos escrever

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

```
long
binomialR2(int n, int k)
{
  if (k == 1) return n;
  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                        0m0.002s
user                        0m0.000s
sys                         0m0.000s

meu_prompt> time ./binomialR2 30 2
binom(30,2)=435
real                        0m0.002s
user                        0m0.000s
sys                         0m0.000s
```

E agora, qual é mais eficiente?

```
meu_prompt> time ./binomialI 30 20
binom(30,20)=30045015
real                        0m0.002s
user                        0m0.000s
sys                         0m0.000s

meu_prompt> time ./binomialR2 30 20
binom(30,20)=30045015
real                        0m0.002s
user                        0m0.000s
sys                         0m0.000s
```

Conclusão

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

Apêndice: Números de Fibonacci

Fonte: http://www.geek.com/geek-cetera/

PF 2.3 S 5.2

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

Números de Fibonacci

$F_0 = 0 \qquad F_1 = 1 \qquad F_n = F_{n-1} + F_{n-2}$

n	0	1	2	3	4	5	6	7	8	9
F_n	0	1	1	2	3	5	8	13	21	34

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);
}
```

fibonacciR(4)

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

Fibonacci iterativo

```
long fibonacciI(int n) {
  long anterior = 0, atual = 1, proximo;
  int i;
  if (n == 0) return 0;
  if (n == 1) return 1;
  for (i = 1; i < n; i++)
    {
      proximo = atual + anterior;
      anterior = atual;
      atual = proximo;
    }
  return atual;
}
```

Qual é mais eficiente?

```
meu_prompt> time ./fibonacciI 10
fibonacci(10)=55
real                  0m0.003s
user                  0m0.000s
sys                   0m0.000s

meu_prompt> time ./fibonacciR 10
fibonacci(10)=55
real                  0m0.003s
user                  0m0.000s
sys                   0m0.000s
```

Qual é mais eficiente?

```
meu_prompt> time ./fibonacciI 20
fibonacci(20) = 6765
real                  0m0.003s
user                  0m0.000s
sys                   0m0.000s

meu_prompt> time ./fibonacciR 20
fibonacci(20) = 6765
real                  0m0.003s
user                  0m0.000s
sys                   0m0.000s
```

Qual é mais eficiente?

```
meu_prompt> time ./fibonacciI 30
fibonacci(30) = 832040
real                  0m0.003s
user                  0m0.000s
sys                   0m0.000s

meu_prompt> time ./fibonacciR 30
fibonacci(30) = 832040
real                  0m0.049s
user                  0m0.044s
sys                   0m0.000s
```

Qual é mais eficiente?

```
meu_prompt> time ./fibonacciI 40
fibonacci(40) = 102334155
real                         0m0.003s
user                         0m0.000s
sys                          0m0.000s

meu_prompt> time ./fibonacciR 40
fibonacci(40) = 102334155
real                         0m4.761s
user                         0m4.756s
sys                          0m0.000s
```

Qual é mais eficiente?

```
meu_prompt> time ./fibonacciI 45
fibonacci(45) = 1134903170
real                         0m0.003s
user                         0m0.000s
sys                          0m0.000s

meu_prompt> time ./fibonacciR 45
fibonacci(45) = 1134903170
real                        0m52.809s
user                        0m52.727s
sys                          0m0.000s
```

fibonacciR(5)

fibonacciR resolve subproblemas muitas vezes.

```
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)
fibonacci(5) = 5.
```

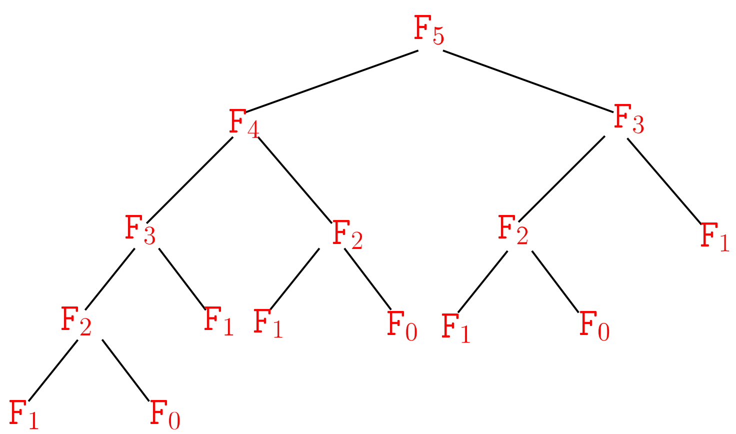
fibonacciR(8)

fibonaccir 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(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.
```

Árvore da recursão

fibonacciR resolve subproblemas muitas vezes.



Consumo de tempo

$T(n)$:= número de somas feitas por $\texttt{fibonacciR}(n)$

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

Consumo de tempo

linha	número de somas
1	$= 0$
2	$= 0$
3	$= T(n-1)$
4	$= T(n-2) + 1$
$T(n)$	$= T(n-1) + T(n-2) + 1$

Recorrência

$T(0) = 0$
$T(1) = 0$
$T(n) = T(n-1) + T(n-2) + 1$ para $n = 2, 3, \ldots$

Uma estimativa para $T(n)$?

Recorrência

$T(0) = 0$
$T(1) = 0$
$T(n) = T(n-1) + T(n-2) + 1$ para $n = 2, 3, \ldots$

Uma estimativa para $T(n)$?

Solução: $T(n) > (3/2)^n$ para $n \geq 6$.

n	0	1	2	3	4	5	6	7	8	9
T_n	0	0	1	2	4	7	12	20	33	54
$(3/2)^n$	1	1.5	2.25	3.38	5.06	7.59	11.39	17.09	25.63	38.44

Recorrência

Prova: $T(6) = 12 > 11.40 > (3/2)^6$ e
$T(7) = 20 > 18 > (3/2)^7$.
Se $n \geq 8$, então

$$\begin{aligned} T(n) &= T(n-1) + T(n-2) + 1 \\ &\overset{\text{hi}}{>} (3/2)^{n-1} + (3/2)^{n-2} + 1 \\ &= (3/2 + 1)\,(3/2)^{n-2} + 1 \\ &> (5/2)\,(3/2)^{n-2} \\ &> (9/4)\,(3/2)^{n-2} \\ &= (3/2)^2 (3/2)^{n-2} \\ &= (3/2)^n\,. \end{aligned}$$

Logo, $T(n) \geq (3/2)^n$. Consumo de tempo é **exponencial.**

Conclusão

O consumo de tempo é da função `fibonacciI(n)` é proporcional a n.

O consumo de tempo da função `fibonacciR` é **exponencial.**

Exercícios

Prove que

$$T(n) = \frac{\phi^{n+1} - \hat{\phi}^{n+1}}{\sqrt{5}} - 1 \quad \text{para } n = 0, 1, 2, \ldots$$

onde

$$\phi = \frac{1+\sqrt{5}}{2} \approx 1{,}61803 \quad \text{e} \quad \hat{\phi} = \frac{1-\sqrt{5}}{2} \approx -0{,}61803.$$

Prove que $1 + \phi = \phi^2$.

Prove que $1 + \hat{\phi} = \hat{\phi}^2$.