Melhores momentos

AULA 2

Conceitos discutidos

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

Desempenho de binomialR1

```
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);
}
```

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.
```

Comparação experimental

```
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
```

Conclusões

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

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

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

Mais conclusões

O consumo de tempo da chamada
`binomialR1(n,k)` é *proporcional a*

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

Quando o valor de `k` é aproximadamente `n`/2
o consumo de tempo da chamada
`binomialR1(n,k)` é *exponencial* pois

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

Binomial mais eficiente

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;
}
```

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 $k - 1$.

AULA 3

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 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:

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) {
```

Fibonacci iterativo

```
long fibonacciI(int n) {
   long anterior = 0, atual = 1, proximo;
   int i;

   if (n == 0) return 0;
   if (n == 1) return 1;
```

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++) {      /*1*/
      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                        0m0.001s
sys                         0m0.001s

meu_prompt> time ./fibonacciR 10
fibonacci(10)=55
real                        0m0.003s
user                        0m0.001s
sys                         0m0.001s
```

Qual é mais eficiente?

```
meu_prompt> time ./fibonacciI 30
fibonacci(30) = 832040
real                        0m0.003s
user                        0m0.001s
sys                         0m0.001s

meu_prompt> time ./fibonacciR 30
fibonacci(30) = 832040
real                        0m0.009s
user                        0m0.007s
sys                         0m0.001s
```

Qual é mais eficiente?

```
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
```

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.
```

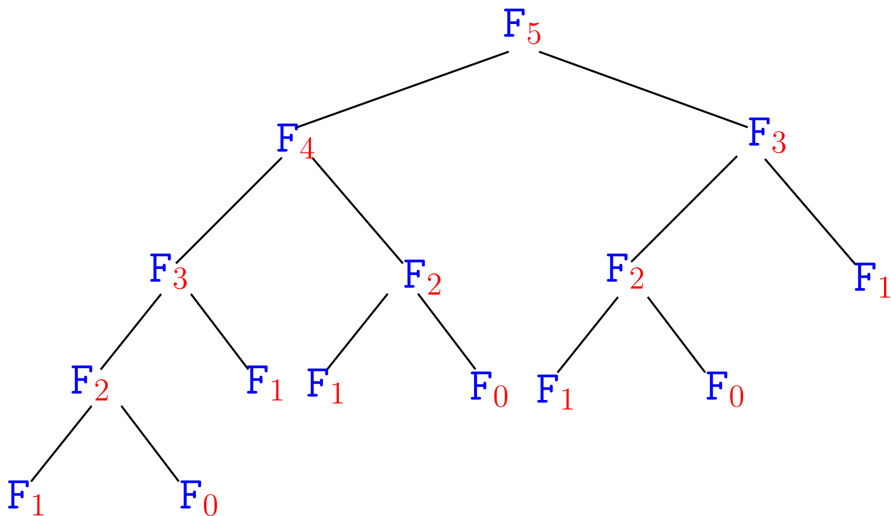
fibonacciR(8)

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 `fibonacciR(n)`

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

Consumo de tempo

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

$$\mathtt{T}(\mathtt{n}) = \mathtt{T}(\mathtt{n} - 1) + \mathtt{T}(\mathtt{n} - 2) + 1$$

Recorrência

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

Uma estimativa para $\mathtt{T}(\mathtt{n})$?

Recorrência

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

Uma estimativa para $T(n)$?

n	0	1	2	3	4	5	6	7	8	9
T_n	0	0	1	2	4	7	12	20	33	54
F_n	0	1	1	2	3	5	8	13	21	34

Recorrência

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

Uma estimativa para $T(n)$?

n	0	1	2	3	4	5	6	7	8	9
T_n	0	0	1	2	4	7	12	20	33	54
F_n	0	1	1	2	3	5	8	13	21	34

$$T(n) = F(n+1) - 1$$

Exercício

Prove que

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

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

Delimitação mais folgada

$$\begin{aligned} \mathtt{T}(\mathtt{n}) &= \text{número de somas feitas por } \mathtt{fibonacciR}(\mathtt{n}) \\ &> (3/2)^{\mathtt{n}} \text{ para } \mathtt{n} \geq 6. \end{aligned}$$

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

Prova por indução

Prova: $\mathtt{T}(6) = 12 > 11.40 > (3/2)^6$ e $\mathtt{T}(7) = 20 > 18 > (3/2)^7$.

Prova por indução

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

$$T(n) = T(n-1) + T(n-2) + 1$$

Prova por indução

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

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

Prova por indução

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 da função `fibonacciI(n)` é proporcional a `n`.

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

Máximo divisor comum

PF 2.3 S 5.1

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

http://www.ime.usp.br/~coelho/mac0122-2012/aulas/mdc/

Divisibilidade

Suponha que $\mathtt{m}$, $\mathtt{n}$ e $\mathtt{d}$ são números inteiros.

Dizemos que $\mathtt{d}$ **divide** $\mathtt{m}$
se $\mathtt{m} = \mathtt{k}\,\mathtt{d}$ para algum número inteiro $\mathtt{k}$.

$\mathtt{d} \mid \mathtt{m}$ é uma abreviatura para "$\mathtt{d}$ divide $\mathtt{m}$"

Divisibilidade

Suponha que $\mathtt{m}, \mathtt{n}$ e $\mathtt{d}$ são números inteiros.

Dizemos que $\mathtt{d}$ **divide** $\mathtt{m}$
se $\mathtt{m} = \mathtt{k}\,\mathtt{d}$ para algum número inteiro $\mathtt{k}$.

$\mathtt{d} \mid \mathtt{m}$ é uma abreviatura para "$\mathtt{d}$ divide $\mathtt{m}$"

Se $\mathtt{d}$ divide $\mathtt{m}$,
então dizemos que $\mathtt{m}$ é um **múltiplo** de $\mathtt{d}$.

Se $\mathtt{d}$ divide $\mathtt{m}$ e $\mathtt{d} > 0$,
então dizemos que $\mathtt{d}$ é um **divisor** de $\mathtt{m}$.

Divisibilidade

Se `d` divide `m` e `d` divide `n`,
então `d` é um **divisor comum** de `m` e `n`.

Exemplos:

os divisores de 30 são: $1, 2, 3, 5, 6, 10, 15$ e 30
os divisores de 24 são: $1, 2, 3, 4, 6, 8, 12$ e 24
os divisores comuns de 30 e 24 são: $1, 2, 3$ e 6

Máximo divisor comum

O **máximo divisor comum** de dois números inteiros `m` e `n`, onde pelo menos um é não nulo, é o maior divisor comum de `m` e `n`.

O máximo divisor comum de `m` e `n` é denotado por $\mathtt{mdc}(\mathtt{m}, \mathtt{n})$.

Máximo divisor comum

O **máximo divisor comum** de dois números inteiros $\mathtt{m}$ e $\mathtt{n}$, onde pelo menos um é não nulo, é o maior divisor comum de $\mathtt{m}$ e $\mathtt{n}$.

O máximo divisor comum de $\mathtt{m}$ e $\mathtt{n}$ é denotado por $\mathtt{mdc}(\mathtt{m}, \mathtt{n})$.

Problema: Dados dois números inteiros não-negativos $\mathtt{m}$ e $\mathtt{n}$, determinar $\mathtt{mdc}(\mathtt{m}, \mathtt{n})$.

Exemplos:
máximo divisor comum de 30 e 24 é 6
máximo divisor comum de 514229 e 317811 é 1
máximo divisor comum de 3267 e 2893 é 11

Solução MAC2166

Recebe números inteiros não-negativos m e n
e devolve $\text{mdc}(m, n)$. Supõe $m, n > 0$.

```
#define min(m,n) ((m) < (n) ? (m) : (n))
```

Solução MAC2166

Recebe números inteiros não-negativos m e n e devolve $\mathtt{mdc}(\mathtt{m}, \mathtt{n})$. Supõe $\mathtt{m}, \mathtt{n} > 0$.

```
#define min(m,n) ((m) < (n) ? (m) : (n))

int mdc(int m, int n) {
   int d = min(m,n);

   while (m % d != 0 || n % d != 0)
       d--;

   return d;
}
```

Consumo de tempo

```
int mdc(int m, int n) {
   int d = min(m,n);
   while (m % d != 0 || n % d != 0)
       d--;
   return d;
}
```

Quantas iterações do `while` faz a função `mdc`?

Em outras palavras,
quantas vezes o comando "`d--`" é executado?

Consumo de tempo

```
int mdc(int m, int n) {
   int d = min(m,n);
   while (m % d != 0 || n % d != 0)
       d--;
   return d;
}
```

Quantas iterações do `while` faz a função `mdc`?

Em outras palavras,
quantas vezes o comando "`d--`" é executado?

A resposta é `min(m,n)-1` ... no **pior caso**.

(Lembre-se que estamos supondo que $m > 0$ e $n > 0$.)

Consumo de tempo

```
int mdc(int m, int n) {
   int d = min(m,n);
   while (m % d != 0 || n % d != 0)
       d--;
   return d;
}
```

São `min(m,n)-1` iterações no **pior caso**.

Por exemplo, para a chamada `mdc`(317811,514229),
a função executará 317811−1 iterações,
pois $\texttt{mdc}(317811, 514229) = 1$,
ou seja, 317811 e 514229 são relativamente primos.

Consumo de tempo

```
int mdc(int m, int n) {
   int d = min(m,n);
   while (m % d != 0 || n % d != 0)
       d--;
   return d;
}
```

Neste caso, costuma-se dizer que
o **consumo de tempo** do algoritmo, no **pior caso**,
é *proporcional a* $\min(\mathtt{m}, \mathtt{n})$, ou ainda, que o consumo
de tempo do algoritmo é da *ordem de* $\min(\mathtt{m}, \mathtt{n})$.

Consumo de tempo

```
int mdc(int m, int n) {
   int d = min(m,n);
   while (m % d != 0 || n % d != 0)
       d--;
   return d;
}
```

Neste caso, costuma-se dizer que
o **consumo de tempo** do algoritmo, no **pior caso**,
é *proporcional a* $\min(m, n)$, ou ainda, que o consumo
de tempo do algoritmo é da *ordem de* $\min(m, n)$.

Isto significa que se o valor de $\min(m, n)$ dobra então
o tempo gasto pela função pode, no **pior caso** dobrar.

Consumo de tempo

```
int mdc(int m, int n) {
   int d = min(m,n);
   while (m % d != 0 || n % d != 0)
       d--;
   return d;
}
```

Neste caso, costuma-se dizer que
o **consumo de tempo** do algoritmo, no **pior caso**,
é *proporcional a* $\min(\mathtt{m}, \mathtt{n})$, ou ainda, que o consumo
de tempo do algoritmo é da *ordem de* $\min(\mathtt{m}, \mathtt{n})$.

A abreviatura de "ordem blá" é $\mathrm{O}(\mathtt{blá})$.

Conclusões

No pior caso, o consumo de tempo da função `mdc` é proporcional a $\min(\mathtt{m}, \mathtt{n})$.

O consumo de tempo da função `mdc` é $O(\min(\mathtt{m}, \mathtt{n}))$.

Se o valor de $\min(\mathtt{m}, \mathtt{n})$ dobra,
o consumo de tempo pode dobrar.