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);
}
```

fatorial(10)

```
fatorial(10)
  fatorial(9)
    fatorial(8)
      fatorial(7)
        fatorial(6)
          fatorial(5)
            fatorial(4)
              fatorial(3)
                fatorial(2)
                  fatorial(1)
                    fatorial(0)
fatorial de 10 e' 3628800.
```

AULA 2

Mais recursão

PF 2.1, 2.2, 2.3   S 5.1

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 binomial_i(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);
}
```

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!

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
```

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

Número de adições

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

Relação de recorrência!

Relação de recorrência

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

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

Número $\mathtt{T}(\mathtt{n},\mathtt{k})$ 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 $\mathtt{T}(\mathtt{n},\mathtt{k})$ de adições feitas pela chamada `binomialr1(n,k)` é

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

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

Quando o valor de $\mathtt{k}$ é aproximadamente $\mathtt{n}/2$

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

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

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

```
long
binomialr2(int n, int k)
{
  if (k == 1) return n;
  return (binomialr2(n-1,k-1)*n) / k;
}
```

A função binomialr3 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 $k-1$.

Argumentos na linha de comando

Argumentos na linha de comando

Quando `main` é chamada, ela recebe dois argumentos:

- `argc` ('c' de *count*) é o número de argumentos que o programa recebeu na linha de comando; e
- `argv[ ]` é um vetor de *strings* contendo cada um dos argumentos.

Por conveção `argv[0]` é o nome do programa que foi chamado. Assim, `argc` é sempre pelo menos 1.

Argumentos na linha de comando

Por exemplo, na chamada

```
meu_prompt> echo Hello World!
```

- `argc` $= 3$
- `argv`[0] = "echo"
- `argv`[1] = "Hello"
- `argv`[2] = "World!"

Argumentos na linha de comando

Na chamada

```
meu_prompt> gcc echo.c -o echo
```

- `argc` $= 4$
- `argv`[0] = "gcc"
- `argv`[1] = "echo.c"
- `argv`[2] = "-o"
- `argv`[3] = "echo"

echo.c

```
#include <stdio.h>

int main(int argc, char *argv[])
{
  int i;

  for (i = 1; i < argc; i++)
    printf("%s ", argv[i]);

  printf("\n");

  return 0;
}
```

echo.java

```
public class echo {

  public static void main(String argv[])
   {
      for (int i=0; i < argv.length; i++)
        System.out.print(argv[i] + " ");

      System.out.print("\n");

      System.exit(0);
    }

}
```