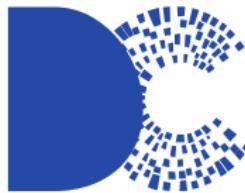




UNIVERSIDADE
FEDERAL DE
SERGIPE



DEPARTAMENTO
DE COMPUTAÇÃO

Relações de recorrência

Projeto e Análise de Algoritmos

Bruno Prado

Departamento de Computação / UFS

Introdução

- ▶ Implementação iterativa do algoritmo de fatorial
 - ▶ Complexidade de tempo

```
1 // Padrão de tipos por tamanho
2 #include <stdint.h>
3 // Fatorial iterativo
4 uint64_t factorial(uint32_t n) {
5     uint64_t r = 1;
6     for(uint32_t i = 2; i <= n; i++)
7         r = r * i;
8     return r;
9 }
```

Introdução

- ▶ Implementação iterativa do algoritmo de fatorial
 - ▶ Complexidade de tempo

```
1 // Padrão de tipos por tamanho
2 #include <stdint.h>
3 // Fatorial iterativo
4 uint64_t factorial(uint32_t n) {
5     uint64_t r = 1;
6     for(uint32_t i = 2; i <= n; i++)
7         r = r * i;
8     return r;
9 }
```

$$fatorial(n) = c_1 + c_2 \times (n - 1) = \Theta(n)$$

Introdução

- ▶ Implementação recursiva do algoritmo de fatorial
 - ▶ Complexidade de tempo

```
1 // Padrão de tipos por tamanho
2 #include <stdint.h>
3 // Fatorial recursivo
4 uint64_t factorial(uint32_t n) {
5     if(n == 0)
6         return 1;
7     else
8         return n * factorial(n - 1);
9 }
```

Introdução

- ▶ Implementação recursiva do algoritmo de fatorial
 - ▶ Complexidade de tempo

```
1 // Padrão de tipos por tamanho
2 #include <stdint.h>
3 // Fatorial recursivo
4 uint64_t factorial(uint32_t n) {
5     if(n == 0)
6         return 1;
7     else
8         return n * factorial(n - 1);
9 }
```

$$fatorial(n) = ???$$

Relações de recorrência

- ▶ O que é uma relação recorrente?
 - ▶ É uma função recursiva que define uma sequência

Relações de recorrência

- ▶ O que é uma relação recorrente?
 - ▶ É uma função recursiva que define uma sequência
- ▶ O que é uma função recursiva?
 - ▶ É uma função descrita em seus próprios termos
 - ▶ Composta por duas partes
 - ▶ Caso base
 - ▶ Recorrência

Relações de recorrência

- ▶ Implementação recursiva do algoritmo de factorial
 - ▶ Caso base ($n = 0$)

```
1 // Padrão de tipos por tamanho
2 #include <stdint.h>
3 // Fatorial recursivo
4 uint64_t factorial(uint32_t n) {
5     if(n == 0)
6         return 1;
7     else
8         return n * factorial(n - 1);
9 }
```



$$fatorial(n) = \begin{cases} 1 & n = 0 \\ fatorial(n-1) + 1 & n > 0 \end{cases}$$

Relações de recorrência

- ▶ Implementação recursiva do algoritmo de factorial
 - ▶ Recorrência ($n > 0$)

```
1 // Padrão de tipos por tamanho
2 #include <stdint.h>
3 // Fatorial recursivo
4 uint64_t factorial(uint32_t n) {
5     if(n == 0)
6         return 1;
7     else
8         return n * factorial(n - 1);
9 }
```



$$fatorial(n) = \begin{cases} 1 & n = 0 \\ fatorial(n-1) + 1 & n > 0 \end{cases}$$

Relações de recorrência

- ▶ Como resolver estas relações de recorrência?
 - ▶ Método de substituição
 - ▶ Método de árvore de recursão
 - ▶ Método mestre

Relações de recorrência

- ▶ Método de substituição

- ▶ $T(n) = \text{fatorial}(n)$

$$T(n) = \begin{cases} 1 & n = 0 \\ T(n-1) + 1 & n > 0 \end{cases}$$

Relações de recorrência

- ▶ Método de substituição
 - ▶ Encontrar um padrão nas substituições

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

Relações de recorrência

- ▶ Método de substituição
 - ▶ Encontrar um padrão nas substituições

$$\begin{aligned}T(n) &= T(n-1) + 1 \\&= [T(n-2) + 1] + 1\end{aligned}$$

Relações de recorrência

- ▶ Método de substituição
 - ▶ Encontrar um padrão nas substituições

$$\begin{aligned}T(n) &= T(n-1) + 1 \\&= [T(n-2) + 1] + 1 \\&= [T(n-3) + 1] + 2\end{aligned}$$

Relações de recorrência

- ▶ Método de substituição
 - ▶ Encontrar um padrão nas substituições

$$\begin{aligned}T(n) &= T(n-1) + 1 \\&= [T(n-2) + 1] + 1 \\&= [T(n-3) + 1] + 2 \\&= [T(n-4) + 1] + 3\end{aligned}$$

Relações de recorrência

- ▶ Método de substituição
 - ▶ Encontrar um padrão nas substituições

$$\begin{aligned}T(n) &= T(n-1) + 1 \\&= [T(n-2) + 1] + 1 \\&= [T(n-3) + 1] + 2 \\&= [T(n-4) + 1] + 3 \\&\quad \vdots \quad \vdots \\&= T(n-k) + k\end{aligned}$$

Relações de recorrência

- ▶ Método de substituição
 - ▶ Aplicar o caso base para encontrar o valor de k

$$T(n) = T(n - k) + k$$

$$T(n - k) = 1 \rightarrow n - k = 0 \rightarrow k = n$$

Relações de recorrência

- ▶ Método de substituição
 - ▶ Aplicar o caso base para encontrar o valor de k

$$T(n) = T(n - k) + k$$

$$T(n - k) = 1 \rightarrow n - k = 0 \rightarrow k = n$$

- ▶ Substituindo na equação

$$\begin{aligned} T(n) &= T(0) + n \\ &= 1 + n \\ &= O(n) \end{aligned}$$

Relações de recorrência

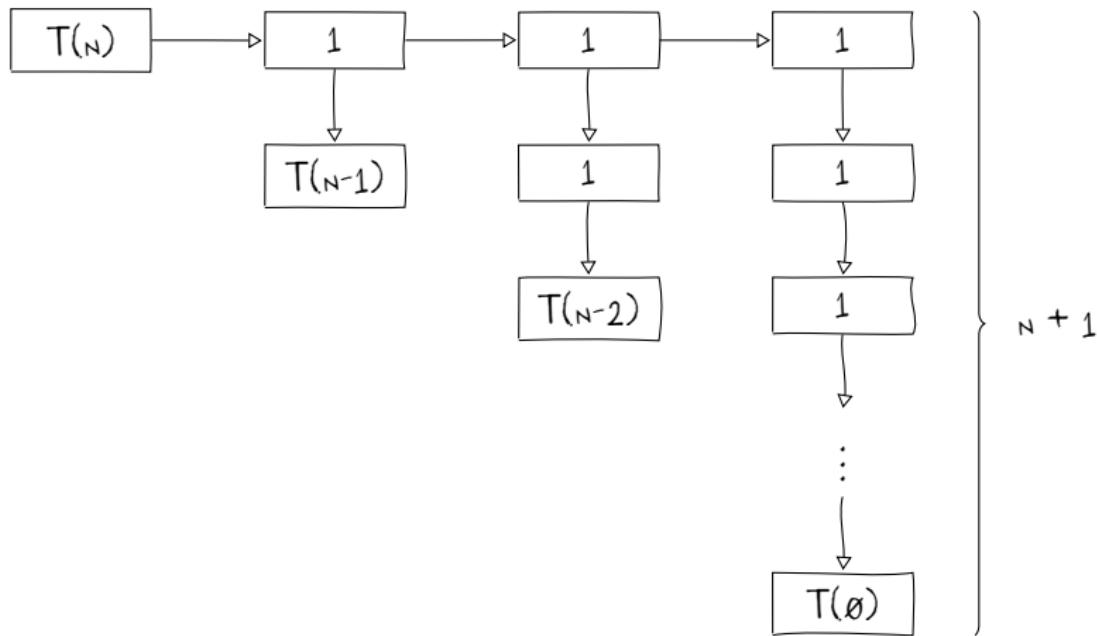
- ▶ Método de árvore de recursão

- ▶ $T(n) = \text{fatorial}(n)$

$$T(n) = \begin{cases} 1 & n = 0 \\ T(n-1) + 1 & n > 0 \end{cases}$$

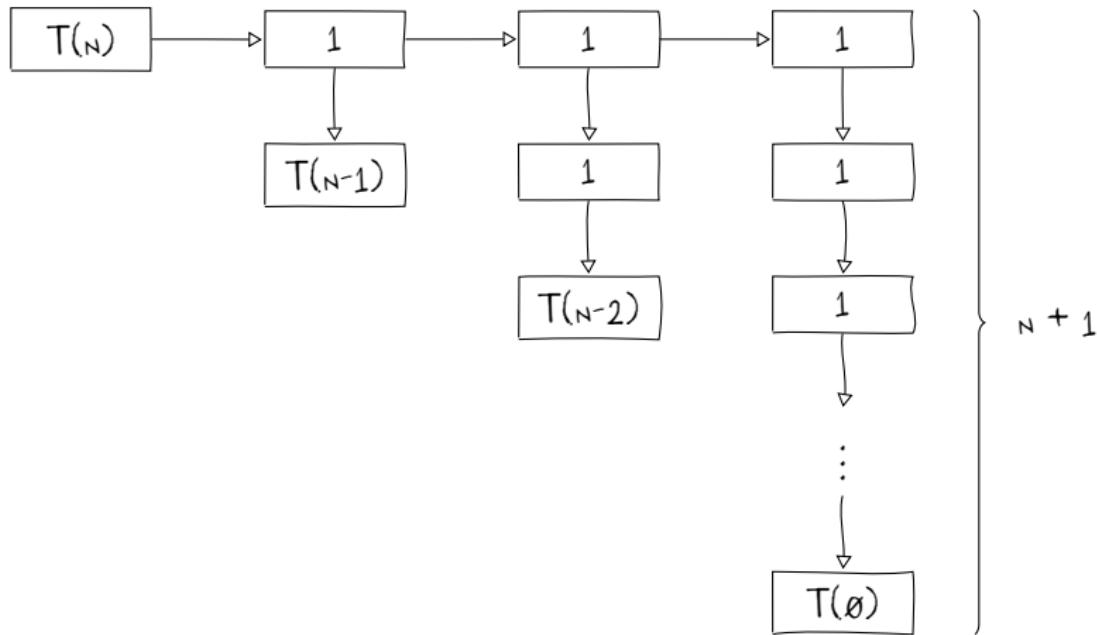
Relações de recorrência

- Método de árvore de recursão
 - Solução gráfica



Relações de recorrência

- Método de árvore de recursão
 - Solução gráfica



$$T(n) = n + 1 = O(n)$$

Relações de recorrência

- ▶ Método mestre
 - ▶ Resolução de relações de recorrência no formato
$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$
 - ▶ $a \geq 1$, número de subproblemas
 - ▶ $b > 1$, tamanho de cada subproblema
 - ▶ $f(n)$ assintoticamente positiva, custo de combinar soluções

Relações de recorrência

► Método mestre

- Caso 1: $f(n) = O(n^{\log_b a - \epsilon})$, com $\epsilon > 0$
- Regra: $T(n) = \Theta(n^{\log_b a})$
- Exemplo: $T(n) = 4T(\frac{n}{2}) + n$
 - $a = 4$ ($a \geq 1$)
 - $b = 2$ ($b > 1$)
 - $f(n) = n$ ($n \rightarrow \infty$, $f(n) > 0$)

$$f(n) = n = O(n^{\log_b a - \epsilon})$$

$$1 = \log_b a - \epsilon$$

$$\epsilon = \log_2 4 - 1$$

$$\epsilon = 1$$



$$T(n) = \Theta(n^2)$$

Relações de recorrência

► Método mestre

- Caso 2: $f(n) = \Theta(n^{\log_b a})$
- Regra: $T(n) = \Theta(n^{\log_b a} \log n)$
- Exemplo: $T(n) = 2T(\frac{n}{2}) + n$
 - $a = 2$ ($a \geq 1$)
 - $b = 2$ ($b > 1$)
 - $f(n) = n$ ($n \rightarrow \infty$, $f(n) > 0$)

$$f(n) = n = \Theta(n^{\log_b a})$$

$$1 = \log_b a$$

$$1 = \log_2 2$$

$$1 = 1$$



$$T(n) = \Theta(n^{\log_2 2} \log n) = \Theta(n \log n)$$

Relações de recorrência

► Método mestre

- Caso 3: $f(n) = \Omega(n^{\log_b a + \epsilon})$, com $\epsilon > 0$, $af(\frac{n}{b}) \leq cf(n)$, $c < 1$ e n suficientemente grande
- Regra: $T(n) = \Theta(f(n))$
- Exemplo: $T(n) = 2T(\frac{n}{2}) + n^2$
 - $a = 2$ ($a \geq 1$)
 - $b = 2$ ($b > 1$)
 - $f(n) = n^2$ ($n \rightarrow \infty$, $f(n) > 0$)

$$f(n) = n^2 = \Omega(n^{\log_b a + \epsilon})$$

$$2 = \log_b a + \epsilon$$

$$\epsilon = 2 - \log_2 2$$

$$\epsilon = 1$$

Relações de recorrência

► Método mestre

- Caso 3: $f(n) = \Omega(n^{\log_b a + \epsilon})$, com $\epsilon > 0$, $af(\frac{n}{b}) \leq cf(n)$, $c < 1$ e n suficientemente grande
- Regra: $T(n) = \Theta(f(n))$
- Exemplo: $T(n) = 2T(\frac{n}{2}) + n^2$
 - $a = 2$ ($a \geq 1$)
 - $b = 2$ ($b > 1$)
 - $f(n) = n^2$ ($n \rightarrow \infty$, $f(n) > 0$)

$$af\left(\frac{n}{b}\right) \leq cf(n) \Rightarrow \frac{n^2}{2} \leq cn^2$$
$$n \rightarrow \infty \Rightarrow c = \frac{1}{2}$$



$$T(n) = \Theta(f(n)) = \Theta(n^2)$$

Relações de recorrência

- ▶ Método mestre
 - ▶ É preciso memorizar os casos
 - ▶ Não resolve todas as recorrências, como no cálculo do fatorial e da sequência de Fibonacci

Relações de recorrência

- ▶ Implementação recursiva do algoritmo de Fibonacci
 - ▶ Caso base ($n \leq 1$)

```
1 // Padrão de tipos por tamanho
2 #include <stdint.h>
3 // Fibonacci recursivo
4 uint64_t fibonacci(uint32_t n) {
5     if(n <= 1)
6         return n;
7     else
8         return fibonacci(n - 1) + fibonacci(n - 2);
9 }
```



$$fibonacci(n) = \begin{cases} 0 & n = 0 \\ 1 & n = 1 \\ fibonacci(n-1) + fibonacci(n-2) & n > 1 \end{cases}$$

Relações de recorrência

- ▶ Implementação recursiva do algoritmo de Fibonacci
 - ▶ Recorrência ($n > 1$)

```
1 // Padrão de tipos por tamanho
2 #include <stdint.h>
3 // Fibonacci recursivo
4 uint64_t fibonacci(uint32_t n) {
5     if(n <= 1)
6         return n;
7     else
8         return fibonacci(n - 1) + fibonacci(n - 2);
9 }
```



$$fibonacci(n) = \begin{cases} 0 & n = 0 \\ 1 & n = 1 \\ fibonacci(n-1) + fibonacci(n-2) & n > 1 \end{cases}$$

Relações de recorrência

- ▶ Implementação recursiva do algoritmo de Fibonacci
 - ▶ Método da substituição

$$\begin{aligned}T(n) &= T(n-1) + T(n-2) \\&= 2T(n-2) + T(n-3) \\&= 3T(n-3) + 2T(n-4) \\&= 5T(n-4) + 3T(n-5) \\&= 8T(n-5) + 5T(n-6) \\&\quad \vdots \quad \vdots \\&= r^{n-1}T(n-1) + r^{n-2}T(n-2)\end{aligned}$$

Relações de recorrência

- ▶ Implementação recursiva do algoritmo de Fibonacci
 - ▶ Método da substituição

$$\begin{aligned}T(n) &= T(n-1) + T(n-2) \\&= 2T(n-2) + T(n-3) \\&= 3T(n-3) + 2T(n-4) \\&= 5T(n-4) + 3T(n-5) \\&= 8T(n-5) + 5T(n-6) \\&\vdots \quad \vdots \\&= r^{n-1}T(n-1) + r^{n-2}T(n-2)\end{aligned}$$

$$r^n = r^{n-1} + r^{n-2} \rightarrow r^2 - r - 1 = 0 \rightarrow r_{1,2} = \frac{1 \pm \sqrt{5}}{2}$$

Recorrência de segunda ordem com raízes distintas

Relações de recorrência

- ▶ Implementação recursiva do algoritmo de Fibonacci
 - ▶ Raízes distintas

$$T(n) = \alpha r_1^n + \beta r_2^n$$

$$T(0) = \alpha \left(\frac{1+\sqrt{5}}{2} \right)^0 + \beta \left(\frac{1-\sqrt{5}}{2} \right)^0 = 0$$

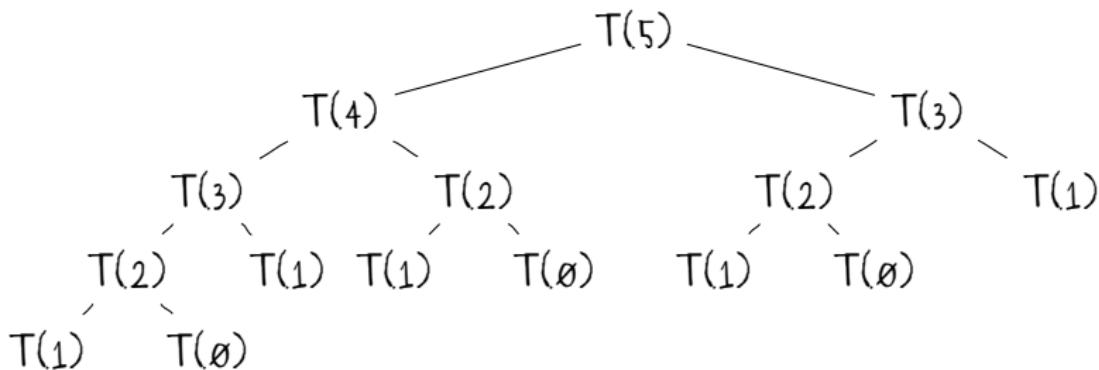
$$T(1) = \alpha \left(\frac{1+\sqrt{5}}{2} \right)^1 + \beta \left(\frac{1-\sqrt{5}}{2} \right)^1 = 1$$

$$T(n) = \frac{1}{\sqrt{5}} \left(\frac{1+\sqrt{5}}{2} \right)^n - \frac{1}{\sqrt{5}} \left(\frac{1-\sqrt{5}}{2} \right)^n$$

$$T(n) \approx \frac{1}{\sqrt{5}} (1,6)^n - \frac{1}{\sqrt{5}} (0,6)^n = O(1,6^n)$$

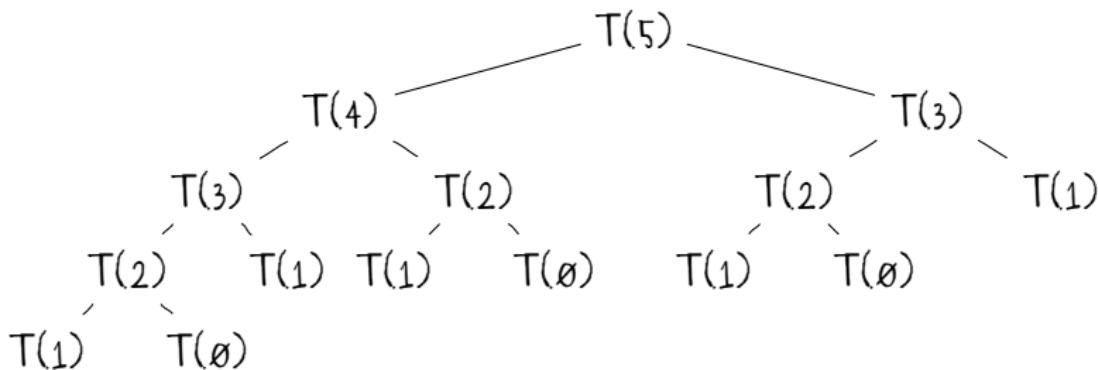
Relações de recorrência

- ▶ Implementação recursiva do algoritmo de Fibonacci
 - ▶ Método da árvore de recursão



Relações de recorrência

- ▶ Implementação recursiva do algoritmo de Fibonacci
 - ▶ Método da árvore de recursão



Árvore com altura n e em cada nó existem 2 filhos
Complexidade é $O(2^n)$

Relações de recorrência

► Implementação iterativa do algoritmo de Fibonacci

```
1 // Padrão de tipos por tamanho
2 #include <stdint.h>
3 // Fibonacci iterativo
4 uint64_t fibonacci(uint32_t n) {
5     uint64_t r = n, tn2 = 0, tn1 = 1;
6     for(uint32_t i = 1; i < n; i++) {
7         r = tn2 + tn1;
8         tn2 = tn1;
9         tn1 = r;
10    }
11    return r;
12 }
```

Relações de recorrência

► Implementação iterativa do algoritmo de Fibonacci

```
1 // Padrão de tipos por tamanho
2 #include <stdint.h>
3 // Fibonacci iterativo
4 uint64_t fibonacci(uint32_t n) {
5     uint64_t r = n, tn2 = 0, tn1 = 1;
6     for(uint32_t i = 1; i < n; i++) {
7         r = tn2 + tn1;
8         tn2 = tn1;
9         tn1 = r;
10    }
11    return r;
12 }
```

$$fibonacci(n) = c_1 \times 3 + c_2 \times (n - 1) = \Theta(n)$$

Exemplo

- ▶ Identifique qual dos algoritmos já visto possui a seguinte recorrência
- ▶ Resolva a recorrência para determinar a complexidade

$$T(n) = \begin{cases} 1 & n = 1 \\ T(n-1) + n & n > 1 \end{cases}$$

Exemplo

- ▶ Método da substituição
 - ▶ Série aritmética

$$\begin{aligned}T(n) &= T(n-1) + n \\&= T(n-2) + (n-1) + n \\&= T(n-3) + (n-2) + (n-1) + n \\&\quad \vdots \quad \vdots \\&= T(1) + \dots + (n-2) + (n-1) + n\end{aligned}$$

Exemplo

- ▶ Método da substituição
 - ▶ Série aritmética

$$\begin{aligned}T(n) &= T(n-1) + n \\&= T(n-2) + (n-1) + n \\&= T(n-3) + (n-2) + (n-1) + n \\&\quad \vdots \quad \vdots \\&= T(1) + \dots + (n-2) + (n-1) + n \\&= \frac{n(n+1)}{2}\end{aligned}$$

Exemplo

- ▶ Método da substituição
 - ▶ Série aritmética

$$\begin{aligned}T(n) &= T(n-1) + n \\&= T(n-2) + (n-1) + n \\&= T(n-3) + (n-2) + (n-1) + n \\&\quad \vdots \quad \vdots \\&= T(1) + \dots + (n-2) + (n-1) + n \\&= \frac{n(n+1)}{2} \\&= \frac{n^2 + n}{2}\end{aligned}$$

Exemplo

- ▶ Método da substituição
 - ▶ Série aritmética

$$\begin{aligned}T(n) &= T(n-1) + n \\&= T(n-2) + (n-1) + n \\&= T(n-3) + (n-2) + (n-1) + n \\&\quad \vdots \quad \vdots \\&= T(1) + \dots + (n-2) + (n-1) + n \\&= \frac{n(n+1)}{2} \\&= \frac{n^2 + n}{2} \\&= O(n^2)\end{aligned}$$

Exercícios

- Resolva as seguintes recorrências para determinar as complexidades

$$\blacktriangleright T(n) = \begin{cases} 1 & n = 1 \\ T\left(\frac{n}{2}\right) + 1 & n > 1 \end{cases}$$

$$\blacktriangleright T(n) = \begin{cases} 1 & n = 1 \\ 4T\left(\frac{n}{2}\right) + n^2 & n > 1 \end{cases}$$