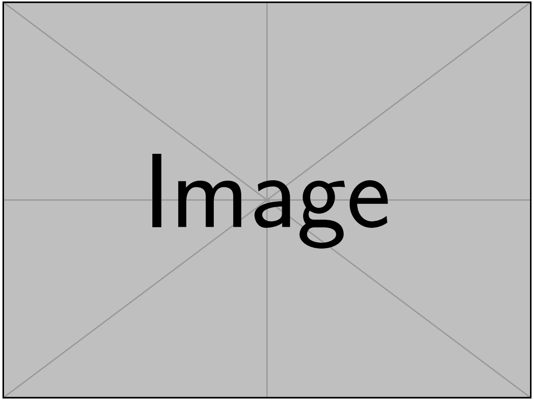
BCC202 - Estruturas de Dados I

Aula 9: Recursividade (Parte II)

Pedro Silva

Universidade Federal de Ouro Preto, UFOP
Departamento de Computação, DECOM
Email: **silvap@ufop.edu.br**

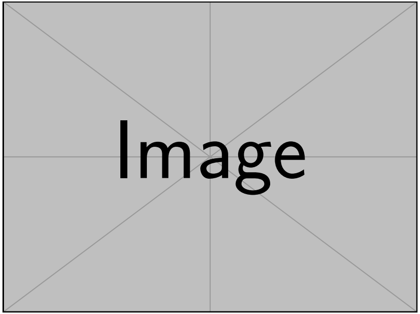
Conteúdo



Image

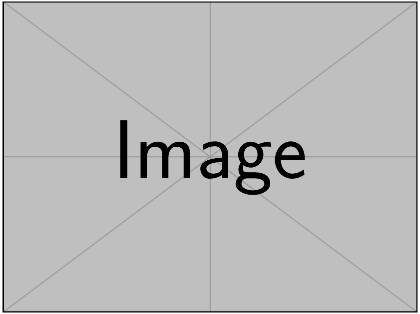
Recursividade

Exemplos de recursividade




```c
void estrela1(int x, int y, int r) {
  if (r>0) {
    estrela1(x - r, y + r, r / 2);
    estrela1(x + r, y + r, r / 2);
    estrela1(x - r, y - r, r / 2);
    estrela1(x + r, y - r, r / 2);
    box(x, y, r);
  }
}
```

Exemplos de recursividade: outra implementação




```c
void estrela2(int x, int y, int r) {
  if (r>1) {
    estrela2(x - r, y + r, r / 2);
    estrela2(x + r, y + r, r / 2);
    estrela2(x - r, y - r, r / 2);
    estrela2(x + r, y - r, r / 2);
  }
  box(x, y, r);
}
```

Qual a melhor implementação?

```c
void estrela1(int x, int y, int r) {
  if (r>0) {
    estrela1(x - r, y + r, r / 2);
    estrela1(x + r, y + r, r / 2);
    estrela1(x - r, y - r, r / 2);
    estrela1(x + r, y - r, r / 2);
    box(x, y, r);
  }
}
```

```c
void estrela2(int x, int y, int r) {
  if (r>1) {
    estrela2(x - r, y + r, r / 2);
    estrela2(x + r, y + r, r / 2);
    estrela2(x - r, y - r, r / 2);
    estrela2(x + r, y - r, r / 2);
  }
  box(x, y, r);
}
```

Análise de Complexidade

Análise de Complexidade - Notação O

- Define-se uma função de complexidade $f(n)$ desconhecida:
  - n mede o tamanho dos argumentos para o procedimento em questão.
- Identifica-se a **equação de recorrência** $T(n)$:
  - Especifica-se $T(n)$ como uma função dos termos anteriores.
  - Especifica-se a **condição de parada** (e.g. $T(1)$).

Recursão de exemplo

```c
void exemplo(int n) {
  int i;
  if(n <= 1)
    printf("%d", n);
  else {
    for(i = 0;i < n; i++)
      printf("%d", n);
    exemplo(n-1);
  }
}
```

Equação de Recorrência: Passo a Passo

- Define-se uma função de complexidade $f(n)$.
- Identifica-se a equação de recorrência $T(n)$:
  - Especifica-se $T(n)$ como uma função dos termos anteriores.
  - Especifica-se a **condição de parada** (e.g. T(1)).

Exemplo de recorrência

```c
void exemplo(int n) {
  int i;
  if(n <= 1)
    printf("%d", n);
  else {
    for(i = 0;i < n; i++)
      printf("%d", n);
    exemplo(n-1);
  }
}
```

Podemos definir a recorrência como?

Exemplo de recorrência

```c
void exemplo(int n) {
  int i;
  if(n <= 1)
    printf("%d", n);
  else {
    for(i = 0;i < n; i++)
      printf("%d", n);
    exemplo(n-1);
  }
}
```

Podemos definir a recorrência como?

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

Resolvendo o exemplo de recorrência

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

Expandindo:

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

Resolvendo o exemplo de recorrência

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

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

$$2T(n) = (n+1) + (n+1) + (n+1) + \ldots + (n+1)$$

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

Resolvendo o exemplo de recorrência

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

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

$$2T(n) = (n+1) + (n+1) + (n+1) + \ldots + (n+1)$$

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

$$\boxed{\text{Logo: } \boldsymbol{T(n) = \frac{n\ (n+1)}{2}}}$$

Resolvendo o exemplo de recorrência

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

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

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

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

$$\boxed{\text{Logo: } \boldsymbol{T(n) = \frac{n\ (n+1)}{2}}} \quad = O(n^2)$$

Exemplos

Exemplo usando fatorial

```
int fatorial(int n) {
  if(n == 1)
    return 1;
  else {
    return n * fatorial(n-1);
  }
}
```

Podemos definir a recorrência como:

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

Fatorial: Análise da função

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

Expandindo:

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

Fatorial: Análise da função

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

Expandindo:

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

$$\boxed{\text{Logo: } \boldsymbol{T(n) = n}}$$

Fatorial: Análise da função

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

Expandindo:

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

$$\boxed{\text{Logo: } \boldsymbol{T(n) = n}} = \boldsymbol{O(n)}$$

Análise de Funções Recursivas

- **Atenção**: lembre-se de que, além da análise de custo de **tempo**, deve-se analisar também o custo de **espaço**.
- Qual a complexidade de espaço da função fatorial (qual o tamanho da pilha de execução)?
  - **Proporcional ao número de chamadas?**

Alguns somatórios úteis

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

$$\sum_{i=0}^{k} a^i = \frac{a^{k+1}-1}{a-1} \quad (a \neq 1)$$

$$\sum_{i=0}^{k} 2^k = 2^{k+1} - 1$$

$$\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$$

$$\sum_{i=0}^{k} \frac{1}{2^i} = 2 - \frac{1}{2^k}$$

Mais um exemplo

Considere a seguinte função:

```c
void func(int n) {
  if (n <= 1)
    printf("*\n");
  else {
    for (int i=0;i<n;i++)
      printf("*");
    printf("\n");
    func(n/3);
  }
}
```

Detalhando a Equação de Recorrência

Qual a equação de recorrência?

Detalhando a Equação de Recorrência

Qual a equação de recorrência?

$$\begin{cases} T(n) = n + T(n/3) \\ T(1) = 1 \end{cases}$$

Detalhando a Equação de Recorrência

Qual a equação de recorrência?

$$\begin{cases} T(n) = n + T(n/3) \\ T(1) = 1 \end{cases}$$

- Resolva a equação de recorrência por expansão.
  - Considere a simplificação de que $\boldsymbol{n}$ seja **sempre divisível por 3**. Ou seja, $\boldsymbol{n = 3^k}$, $\boldsymbol{k >= 0}$.
  - Dica: Somatório de uma PG finita $= \boldsymbol{a_1(1 - q^n)/(1 - q)}$, onde n = número de termos da PG.

Resolvendo a Equação de Recorrência

$$\begin{cases} T(n) = n + T(n/3) \\ T(1) = 1 \end{cases}$$

Resolvendo por expansão:

$$\begin{aligned} T(n) &= n + T(n/3) \\ T(n/3) &= n/3 + T(n/3/3) \\ T(n/3/3) &= n/3/3 + T(n/3/3/3) \\ &\vdots \\ T(n/3/3.../3) &= n/3/3/.../3 + T(n/3/3/.../3) \end{aligned}$$

Resolvendo a Equação de Recorrência

$$\begin{cases} T(n) = n + T(n/3) \\ T(1) = 1 \end{cases}$$

Resolvendo por expansão:

$$\begin{aligned} T(n) &= n + T(n/3) \\ &= n + n/3 + T(n/3/3) \\ &= n + n/3 + n/3/3 + T(n/3/3/3) \\ &\vdots \\ &= n + n/3 + n/3/3 + \ldots + n/3/3/\ldots/3 + T(n/3/3/\ldots/3) \end{aligned}$$

Resolvendo a Equação de Recorrência

Pela expansão chegamos a:

$$\begin{aligned} T(n) &= n + n/3 + n/3/3 + \ldots + n/3/3/\ldots/3 + T(n/3/3/\ldots/3) \\ &= \frac{n}{3^0} + \frac{n}{3^1} + \frac{n}{3^2} + \ldots + \frac{n}{3^{k-1}} + \frac{n}{3^k} \end{aligned}$$

Considerando que:

$$1 = \frac{n}{3^k} \quad \Rightarrow \quad n = 3^k$$

Tem-se que:

$$T\left(\frac{n}{3^k}\right) = T(1)$$

Resolvendo a Equação de Recorrência

Considerando que $T\left(\frac{n}{3^k}\right) = T(1)$, temos:

$$T(n) = \sum_{i=0}^{k-1}\left(\frac{n}{3^i}\right) + T(1) \quad = \quad n\sum_{i=0}^{k-1}(1/3^i) + 1$$

Mas, $\sum_{i=0}^{k-1}\left(\frac{1}{3^i}\right)$ é uma PG finita, com $a_1 = 1$, $q = \frac{1}{3}$ e $n = 3^k$.

Resolvendo a Equação de Recorrência

Dado: $T(n) = \sum_{i=0}^{k-1}(1/3^i) + 1$

Sabe-se que:

- Somatório de uma PG finita é $\sum_{i=0}^{k-1} q^i = a_1 \frac{1-q^n}{1-q}$.
- Com $a_1 = 1$, $q = \frac{1}{3}$ e $n = 3^k$.

Tem-se:

$$T(n) = n\,\frac{1-\left(\frac{1}{3}\right)^k}{1-\frac{1}{3}} + 1$$

Resolvendo a Equação de Recorrência

Dado: $T(n) = \sum_{i=0}^{k-1} (1/3^i) + 1$

Sabe-se que:

- Somatório de uma PG finita é $\sum_{i=0}^{k-1} q^i = a_1 \frac{1-q^n}{1-q}$.
- Com $a_1 = 1$, $q = \frac{1}{3}$ e $n = 3^k$.

Tem-se:

$$T(n) = n\,\frac{1-\left(\frac{1}{3}\right)^k}{1-\frac{1}{3}} + 1$$

$$= n\,\frac{1-\frac{1}{n}}{1-\frac{1}{3}} + 1$$

Resolvendo a Equação de Recorrência

Dado: $T(n) = \sum_{i=0}^{k-1} (1/3^i) + 1$

Sabe-se que:

- Somatório de uma PG finita é $\sum_{i=0}^{k-1} q^i = a_1 \frac{1-q^n}{1-q}$.
- Com $a_1 = 1$, $q = \frac{1}{3}$ e $n = 3^k$.

Tem-se:

$$
\begin{aligned}
T(n) &= n \, \frac{1 - \left(\frac{1}{3}\right)^k}{1 - \frac{1}{3}} + 1 \\
&= n \, \frac{1 - \frac{1}{n}}{1 - \frac{1}{3}} + 1 \\
&= \frac{n-1}{\frac{2}{3}} + 1
\end{aligned}
$$

Resolvendo a Equação de Recorrência

Dado: $T(n) = \sum_{i=0}^{k-1} (1/3^i) + 1$

Sabe-se que:

- Somatório de uma PG finita é $\sum_{i=0}^{k-1} q^i = a_1 \frac{1-q^n}{1-q}$.
- Com $a_1 = 1$, $q = \frac{1}{3}$ e $n = 3^k$.

Tem-se:

$$
\begin{aligned}
T(n) &= n \, \frac{1 - \left(\frac{1}{3}\right)^k}{1 - \frac{1}{3}} + 1 \\
&= n \, \frac{1 - \frac{1}{n}}{1 - \frac{1}{3}} + 1 \\
&= \frac{n-1}{\frac{2}{3}} + 1 \\
&= \frac{3n}{2} - \frac{1}{2}
\end{aligned}
$$

Resolvendo a Equação de Recorrência

Dado: $T(n) = \sum_{i=0}^{k-1} (1/3^i) + 1$

Sabe-se que:

- Somatório de uma PG finita é $\sum_{i=0}^{k-1} q^i = a_1 \frac{1 - q^n}{1 - q}$.
- Com $a_1 = 1$, $q = \frac{1}{3}$ e $n = 3^k$.

Tem-se:

$$
\begin{aligned}
T(n) &= n \, \frac{1 - \left(\frac{1}{3}\right)^k}{1 - \frac{1}{3}} + 1 \\
&= n \, \frac{1 - \frac{1}{n}}{1 - \frac{1}{3}} + 1 \\
&= \frac{n - 1}{\frac{2}{3}} + 1 \\
&= \frac{3n}{2} - \frac{1}{2}
\end{aligned}
$$

Portanto, $\boldsymbol{T(n) = O(n)}$

Algumas recorrências básicas

$$\begin{cases} T(n) = T(\frac{n}{2}) + 1 & (n \geq 2) \\ T(1) = 0 & (n = 1) \end{cases}$$

Algumas recorrências básicas

$$\begin{cases} T(n) = T(\frac{n}{2}) + 1 & (n \geq 2) \\ T(1) = 0 & (n = 1) \end{cases}$$

Vamos supor que:

$$n = 2^k \Rightarrow k = \log n$$

Algumas recorrências básicas

$$\begin{cases} T(n) = T(\frac{n}{2}) + 1 & (n \geq 2) \\ T(1) = 0 & (n = 1) \end{cases}$$

Vamos supor que:

$$n = 2^k \Rightarrow k = \log n$$

Resolvendo por expansão temos:

$$\begin{aligned} T(n) &= T(2^{k-1}) + 1 \\ &= (T(2^{k-2}) + 1) + 1 \\ &= (T(2^{k-3}) + 1) + 1 + 1 \\ &\vdots \\ &= (T(2) + 1) + 1 + \ldots + 1 \\ &= (T(1) + 1) + 1 + \ldots + 1 \\ &= 0 + \underbrace{1 + 1 + \ldots + 1}_{k} \\ &= k \end{aligned}$$

Algumas recorrências básicas

$$\begin{cases} T(n) = T(\frac{n}{2}) + 1 & (n \geq 2) \\ T(1) = 0 & (n = 1) \end{cases}$$

Vamos supor que:

$$n = 2^k \Rightarrow k = \log n$$

Resolvendo por expansão temos:

$$\begin{aligned} T(n) &= T(2^{k-1}) + 1 \\ &= (T(2^{k-2}) + 1) + 1 \\ &= (T(2^{k-3}) + 1) + 1 + 1 \\ &\vdots \\ &= (T(2) + 1) + 1 + \ldots + 1 \\ &= (T(1) + 1) + 1 + \ldots + 1 \\ &= 0 + \underbrace{1 + 1 + \ldots + 1}_{k} \\ &= k \end{aligned}$$

$$T(n) = \log n \Rightarrow O(\log n)$$

Outro exemplo de pesquisa

```c
int pesquisa(int* vetor, int x, int ini, int fim) {
  if (ini > fim)
    return -1;

  int meio = (ini + fim) / 2;

  if (vetor[meio] == x)
    return meio;
  else if (vetor[meio] < x)
    return pesquisa(vetor, x, meio + 1, fim);
  else
    return pesquisa(vetor, x, ini, meio - 1);
}
```

Tarefa 1

- Função recursiva que calcula a potência de um número:
  - Qual a condição de parada.

Tarefa 1

- Função recursiva que calcula a potência de um número:
  - Qual a condição de parada.
- Apresente a equação de recorrência e resolva-a.

Tarefa 1

- Função recursiva que calcula a potência de um número:
  - Qual a condição de parada.
- Apresente a equação de recorrência e resolva-a.
- Qual a complexidade de tempo desta função?

Tarefa 1 - Resolução

```c
int pot(int base, int exp) {
  if (exp == 0)
    return 1;
  /* else */
  return (base * pot(base, exp - 1));
}
```

Análise de complexidade:

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

Tarefa 1 - Resolução

```c
int pot(int base, int exp) {
  if (exp == 0)
    return 1;
  /* else */
  return (base * pot(base, exp - 1));
}
```

Análise de complexidade:

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

$$O(n)$$

Tarefa 2

- Implementar uma função recursiva para computar o valor de 2^n.
  - Qual a condição de parada.

Tarefa 2

- Implementar uma função recursiva para computar o valor de 2^n.
  - Qual a condição de parada.
- Apresente a equação de recorrência e resolva-a.

Tarefa 2

- Implementar uma função recursiva para computar o valor de 2^n.
  - Qual a condição de parada.
- Apresente a equação de recorrência e resolva-a.
- Qual a complexidade de tempo desta função?

Tarefa 2 - Resolução

```c
int pot2(int exp) {
  if (exp == 0)
    return 1;
  /* else */
  return 2 * pot2(base, exp - 1);
}
```

Análise de complexidade:

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

Tarefa 2 - Resolução

```c
int pot2(int exp) {
  if (exp == 0)
    return 1;
  /* else */
  return 2 * pot2(base, exp - 1);
}
```

Análise de complexidade:

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

$$O(n)$$

Tarefa 3

- Indique e resolva a equação de recorrência da função abaixo. Qual a complexidade assintótica?

```c
void f(int a, int b) {
  if (b==0)
    return a;
  return f(a + a, b - 1);
}
```

- Indique o algoritmo e a equação de recorrência do algoritmo de Euclides (calcula o MDC - Máximo Divisor Comum - de dois números a e b).

Conclusão

Conclusão

- Revisão de conceitos de **recursividade**.
- Noção geral sobre complexidade de funções recursivas por meio de **equações de recorrência**.

Listas.

Bibliografia

Bibliografia

Os conteúdos deste material, incluindo 4-tad/figs/, textos e códigos, foram extraídos ou adaptados do livro-texto indicado a seguir:

Celes, Waldemar and Cerqueira, Renato and Rangel, José
Introdução a Estruturas de Dados com Técnicas de Programação em C.
Elsevier Brasil, 2016.
ISBN 978-85-352-8345-7.

Exercícios

Exercício 01

- Crie uma função recursiva que calcula a média de um vetor.
  - Qual a condição de parada?
  - Qual a complexidade de sua função? Apresente a equação de recorrência e resolva-a.