Projeto e Análise de Algoritmos
Recorrências e Algoritmos Recursivos

Nelson Cruz Sampaio Neto
nelsonneto@ufpa.br

Universidade Federal do Pará
Instituto de Ciências Exatas e Naturais
Programa de Pós-Graduação em Ciência da Computação

16 de março de 2023

Sequência

- **Definição**: Sequência é uma lista (**finita ou não**) de objetos em uma **ordem** específica. Exemplo: 1, 0, 0, 1, 1, 0, 0, 1.
- Existe um "primeiro elemento", "um segundo elemento", ...
- A sequência do quadrado dos números inteiros e positivos 1, 4, 9, 16, ... é infinita e pode ser definida por n^2, onde $n \geq 1$.
- A sequência exponencial 2, 4, 8, 16, ..., 256 é finita e pode ser definida por 2^n, onde $1 \leq n \leq 8$.
- Sequência de caracteres (letras e outros símbolos) é chamada de cadeia ou *string*. Exemplo: "abacate".

Recursão

- Às vezes é difícil definir um objeto explicitamente. Contudo, pode ser fácil defini-lo em termos dele mesmo.
- Esse processo é chamado de **recursão**, e pode ser usado para definir sequências, funções e conjuntos.
- Exemplo: A sequência de potência de 2 é dada por $T(n) = 2^n$ para $n \geq 0$. Outra forma de definir essa sequência é explicitar seu primeiro termo e estabelecer uma regra para encontrar um termo a partir do anterior, ou seja,

  $T(0) = 1$ e

  $T(n) = 2 \cdot T(n-1)$ para $n \geq 1$.

  Essa definição é chamada de **recursividade** e a regra recursiva de **relação de recorrência**.

Definição recursiva

- Uma sequência T é definida recursivamente nomeando-se seu primeiro valor (ou alguns primeiros valores) e definindo-se seus valores posteriores em termos dos valores anteriores.

- **Exemplo**: Qual seria a definição recursiva da sequência infinita: 3, 7, 11, 15, 19, 23, ...?

Definição recursiva

- Uma sequência T é definida recursivamente nomeando-se seu primeiro valor (ou alguns primeiros valores) e definindo-se seus valores posteriores em termos dos valores anteriores.

- **Exemplo**: Qual seria a definição recursiva da sequência infinita: 3, 7, 11, 15, 19, 23, ...?

  $T(1) = 3$ e
  $T(n) = T(n-1) + 4$ para $n \geq 2$.

Definição recursiva

- Uma sequência T é definida recursivamente nomeando-se seu primeiro valor (ou alguns primeiros valores) e definindo-se seus valores posteriores em termos dos valores anteriores.

- **Exemplo**: Qual seria a definição recursiva da sequência infinita: 3, 7, 11, 15, 19, 23, ...?

  $T(1) = 3$ e
  $T(n) = T(n-1) + 4$ para $n \geq 2$.

- **Exemplo**: Qual seria a definição recursiva da sequência finita: 5, 10, 20, 40, 80, 160?

Definição recursiva

- Uma sequência T é definida recursivamente nomeando-se seu primeiro valor (ou alguns primeiros valores) e definindo-se seus valores posteriores em termos dos valores anteriores.

- **Exemplo**: Qual seria a definição recursiva da sequência infinita: 3, 7, 11, 15, 19, 23, ...?

  $T(1) = 3$ e
  $T(n) = T(n-1) + 4$ para $n \geq 2$.

- **Exemplo**: Qual seria a definição recursiva da sequência finita: 5, 10, 20, 40, 80, 160?

  $T(1) = 5$ e
  $T(n) = 2T(n-1)$ para $2 \leq n \leq 6$.

Definição recursiva

- **Exemplo**: Seja T uma sequência definida por:

  $T(n) = T(n-1) - T(n-2)$ para $n \geq 2$.

  Considerando que $T(0) = 3$ e $T(1) = 5$, calcule $T(2)$ e $T(3)$.

Definição recursiva

- **Exemplo**: Seja T uma sequência definida por:

  $T(n) = T(n-1) - T(n-2)$ para $n \geq 2$.

  Considerando que $T(0) = 3$ e $T(1) = 5$, calcule $T(2)$ e $T(3)$.

  $T(2) = T(1) - T(0) = 5 - 3 = 2$.

  $T(3) = T(2) - T(1) = 2 - 5 = -3$.

Definição recursiva

- **Exemplo**: Seja T uma sequência definida por:

  $$T(n) = T(n-1) - T(n-2) \text{ para } n \geq 2.$$

  Considerando que $T(0) = 3$ e $T(1) = 5$, calcule $T(2)$ e $T(3)$.

  $$T(2) = T(1) - T(0) = 5 - 3 = 2.$$

  $$T(3) = T(2) - T(1) = 2 - 5 = -3.$$

- **Exemplo**: Determine se a sequência T definida por $3n$, para $n \geq 0$, também pode ser definida recursivamente por

  $$T(0) = 0,\ T(1) = 3 \text{ e}$$

  $$T(n) = 2T(n-1) - T(n-2) \text{ para } n \geq 2.$$

Definição recursiva

- **Hipótese Indutiva**: $T(n) = 3n$.

- **Hipótese Indutiva**: $T(n) = 3n$.

$$T(0) = 3 * 0 = 0 \quad \text{OK!}$$

Definição recursiva

- **Hipótese Indutiva**: $T(n) = 3n$.

$T(0) = 3 * 0 = 0$ OK!

$T(1) = 3 * 1 = 3$ OK!

Definição recursiva

- **Hipótese Indutiva**: $T(n) = 3n$.

  $T(0) = 3 * 0 = 0$ OK!

  $T(1) = 3 * 1 = 3$ OK!

  Agora, queremos demonstrar que $T(n+1) = 3(n+1)$.

Definição recursiva

- **Hipótese Indutiva**: $T(n) = 3n$.

  $T(0) = 3 * 0 = 0$ OK!

  $T(1) = 3 * 1 = 3$ OK!

  Agora, queremos demonstrar que $T(n+1) = 3(n+1)$.

  Da relação de recorrência e usando a hipótese indutiva:

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

Definição recursiva

- **Hipótese Indutiva**: $T(n) = 3n$.

$T(0) = 3 * 0 = 0$ OK!

$T(1) = 3 * 1 = 3$ OK!

Agora, queremos demonstrar que $T(n+1) = 3(n+1)$.

Da relação de recorrência e usando a hipótese indutiva:

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

Definição recursiva

- **Hipótese Indutiva**: $T(n) = 3n$.

  $T(0) = 3 * 0 = 0$ OK!

  $T(1) = 3 * 1 = 3$ OK!

  Agora, queremos demonstrar que $T(n+1) = 3(n+1)$.

  Da relação de recorrência e usando a hipótese indutiva:

  $$T(n+1) = 2T(n+1-1) - T(n+1-2)$$
  $$T(n+1) = 2T(n) - T(n-1)$$
  $$T(n+1) = 2(3n) - 3(n-1)$$

Definição recursiva

- **Hipótese Indutiva**: $T(n) = 3n$.

  $T(0) = 3 * 0 = 0 \quad \text{OK!}$

  $T(1) = 3 * 1 = 3 \quad \text{OK!}$

  Agora, queremos demonstrar que $T(n+1) = 3(n+1)$.

  Da relação de recorrência e usando a hipótese indutiva:

$$
\begin{aligned}
T(n+1) &= 2T(n+1-1) - T(n+1-2) \\
T(n+1) &= 2T(n) - T(n-1) \\
T(n+1) &= 2(3n) - 3(n-1) \\
T(n+1) &= 6n - 3n + 3
\end{aligned}
$$

Definição recursiva

- **Hipótese Indutiva**: $T(n) = 3n$.

  $T(0) = 3 * 0 = 0$ OK!

  $T(1) = 3 * 1 = 3$ OK!

  Agora, queremos demonstrar que $T(n+1) = 3(n+1)$.

  Da relação de recorrência e usando a hipótese indutiva:

  $$T(n+1) = 2T(n+1-1) - T(n+1-2)$$
  $$T(n+1) = 2T(n) - T(n-1)$$
  $$T(n+1) = 2(3n) - 3(n-1)$$
  $$T(n+1) = 6n - 3n + 3$$
  $$T(n+1) = 3n + 3$$

Definição recursiva

- **Hipótese Indutiva**: $T(n) = 3n$.

  $T(0) = 3 * 0 = 0$ OK!

  $T(1) = 3 * 1 = 3$ OK!

  Agora, queremos demonstrar que $T(n+1) = 3(n+1)$.

  Da relação de recorrência e usando a hipótese indutiva:

  $$T(n+1) = 2T(n+1-1) - T(n+1-2)$$
  $$T(n+1) = 2T(n) - T(n-1)$$
  $$T(n+1) = 2(3n) - 3(n-1)$$
  $$T(n+1) = 6n - 3n + 3$$
  $$T(n+1) = 3n + 3$$
  $$T(n+1) = 3(n+1) \quad \text{c.q.d}$$

Resolvendo relações de recorrência

- Nem sempre a tarefa de deduzir a função que define uma sequência é simples. Às vezes, é mais fácil enxergar sua definição recursiva e, a partir dela, obter a função requerida.

- Então, resolver uma relação de recorrência significa encontrar uma **fórmula fechada** (ou explícita).

- Ou seja, encontrar uma função onde pode-se substituir uma variável e encontrar o valor pretendido.

- **Exemplo**: Resolva a relação de recorrência:

  $T(n) = 2T(n-1)$ para $n \geq 2$ e $T(1) = 2$.

  A resposta é a fórmula fechada: $T(n) = 2^n$. Como?

Resolvendo relações de recorrência

- Uma das técnicas para resolver relação de recorrência tem três passos: expandir, conjecturar e verificar.

Resolvendo relações de recorrência

- Uma das técnicas para resolver relação de recorrência tem três passos: expandir, conjecturar e verificar.

- **Expandir**:
  $k = 1$: $T(n) = 2T(n-1)$

Resolvendo relações de recorrência

- Uma das técnicas para resolver relação de recorrência tem três passos: expandir, conjecturar e verificar.

- **Expandir**:
  $k = 1$: $T(n) = 2T(n-1)$
  $k = 2$: $T(n) = 2[2T(n-2)] = 4T(n-2)$

Resolvendo relações de recorrência

- Uma das técnicas para resolver relação de recorrência tem três passos: expandir, conjecturar e verificar.

- **Expandir**:
  $k = 1$: $T(n) = 2T(n-1)$
  $k = 2$: $T(n) = 2[2T(n-2)] = 4T(n-2)$
  $k = 3$: $T(n) = 4[2T(n-3)] = 8T(n-3)$

Resolvendo relações de recorrência

- Uma das técnicas para resolver relação de recorrência tem três passos: expandir, conjecturar e verificar.

- **Expandir**:
  $k = 1$: $T(n) = 2T(n-1)$
  $k = 2$: $T(n) = 2[2T(n-2)] = 4T(n-2)$
  $k = 3$: $T(n) = 4[2T(n-3)] = 8T(n-3)$

- **Conjecturar**: Após k expansões, temos $T(n) = 2^k T(n-k)$.

  Observando a expansão, ela irá parar quando $k = n - 1$, isso porque a base da recursividade é definida para 1.

  Logo, $T(n) = 2^n$.

Resolvendo relações de recorrência

- Uma das técnicas para resolver relação de recorrência tem três passos: expandir, conjecturar e verificar.

- **Expandir**:
  $k = 1$: $T(n) = 2T(n-1)$
  $k = 2$: $T(n) = 2[2T(n-2)] = 4T(n-2)$
  $k = 3$: $T(n) = 4[2T(n-3)] = 8T(n-3)$

- **Conjecturar**: Após k expansões, temos $T(n) = 2^k T(n-k)$.

  Observando a expansão, ela irá parar quando $k = n - 1$, isso porque a base da recursividade é definida para 1.

  Logo, $T(n) = 2^n$.

- **Verificar**: Provar a conjectura via indução matemática.

Resolvendo relações de recorrência

- Para verificar a solução da fórmula fechada precisamos agora demonstrar sua validade por indução matemática em n.

- Para verificar a solução da fórmula fechada precisamos agora demonstrar sua validade por indução matemática em n.

  Hipótese Indutiva: $T(n) = 2^n$.

- Para verificar a solução da fórmula fechada precisamos agora demonstrar sua validade por indução matemática em n.

  Hipótese Indutiva: $T(n) = 2^n$.

  $T(1) = 2^1 = 2$ OK!

- Para verificar a solução da fórmula fechada precisamos agora demonstrar sua validade por indução matemática em n.

Hipótese Indutiva: $T(n) = 2^n$.

$T(1) = 2^1 = 2$ OK!

Agora, queremos demonstrar que $T(n+1) = 2^{n+1}$.

- Para verificar a solução da fórmula fechada precisamos agora demonstrar sua validade por indução matemática em n.

  Hipótese Indutiva: $T(n) = 2^n$.

  $T(1) = 2^1 = 2$ OK!

  Agora, queremos demonstrar que $T(n+1) = 2^{n+1}$.

  Da relação de recorrência e usando a hipótese indutiva:

- Para verificar a solução da fórmula fechada precisamos agora demonstrar sua validade por indução matemática em n.

  Hipótese Indutiva: $T(n) = 2^n$.

  $T(1) = 2^1 = 2$ OK!

  Agora, queremos demonstrar que $T(n+1) = 2^{n+1}$.

  Da relação de recorrência e usando a hipótese indutiva:

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

- Para verificar a solução da fórmula fechada precisamos agora demonstrar sua validade por indução matemática em n.

  Hipótese Indutiva: $T(n) = 2^n$.

  $T(1) = 2^1 = 2$ OK!

  Agora, queremos demonstrar que $T(n+1) = 2^{n+1}$.

  Da relação de recorrência e usando a hipótese indutiva:

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

- Para verificar a solução da fórmula fechada precisamos agora demonstrar sua validade por indução matemática em n.

  Hipótese Indutiva: $T(n) = 2^n$.

  $T(1) = 2^1 = 2$ OK!

  Agora, queremos demonstrar que $T(n+1) = 2^{n+1}$.

  Da relação de recorrência e usando a hipótese indutiva:

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

- Para verificar a solução da fórmula fechada precisamos agora demonstrar sua validade por indução matemática em n.

  Hipótese Indutiva: $T(n) = 2^n$.

  $T(1) = 2^1 = 2$ OK!

  Agora, queremos demonstrar que $T(n+1) = 2^{n+1}$.

  Da relação de recorrência e usando a hipótese indutiva:

  $$T(n+1) = 2T(n+1-1)$$
  $$T(n+1) = 2T(n)$$
  $$T(n+1) = 2(2^n)$$
  $$T(n+1) = 2^{n+1} \quad \text{c.q.d}$$

Exemplo

- **Exemplo**: Resolva a seguinte relação de recorrência:

  $T(n) = T(n-1) + n - 1$ para $n \geq 2$ e $T(1) = 0$.

- A saber, somatórios de $(n - i)$:

  $\sum_{i=0}^{n-1}(n-i) = \frac{n(n+1)}{2}$

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

Exemplo

- Passo base: $T(1) = 0$

- **Expandir**:

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

Exemplo

- Passo base: $T(1) = 0$

- **Expandir**:

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

  $k = 2$: $T(n) = [T(n-2) + n - 1 - 1] + n - 1 = T(n-2) + n - 2 + n - 1$

Exemplo

- Passo base: $T(1) = 0$
- **Expandir**:
  $k = 1$: $T(n) = T(n-1) + n - 1$
  $k = 2$: $T(n) = [T(n-2) + n - 1 - 1] + n - 1 = T(n-2) + n - 2 + n - 1$
  $k = 3$:
  $T(n) = [T(n-3) + n - 2 - 1] + n - 2 + n - 1 = T(n-3) + n - 3 + n - 2 + n - 1$

Exemplo

- Passo base: $T(1) = 0$
- **Expandir**:
  $k = 1$: $T(n) = T(n-1) + n - 1$
  $k = 2$: $T(n) = [T(n-2) + n - 1 - 1] + n - 1 = T(n-2) + n - 2 + n - 1$
  $k = 3$:
  $T(n) = [T(n-3) + n - 2 - 1] + n - 2 + n - 1 = T(n-3) + n - 3 + n - 2 + n - 1$
- **Conjecturar**: Após k expansões, tem-se:

  $$T(n) = T(n-k) + \sum_{i=1}^{k}(n-i).$$

  Observando a expansão, ela irá parar quando $k = n - 1$, isso porque a base da recursividade é definida para 1.

  Logo, $T(n) = T(1) + \sum_{i=1}^{n-1}(n-i) = \frac{n(n-1)}{2}$.

Exemplo

- Suponha que uma pessoa abriu uma conta poupança e nela depositou R$ 100,00. Sabendo que o rendimento da poupança é de 3% ao ano, encontre uma fórmula fechada que expresse quanto essa pessoa terá na conta depois de um determinado número de anos. Por exemplo, 30 anos.

Exemplo

- Suponha que uma pessoa abriu uma conta poupança e nela depositou R$ 100,00. Sabendo que o rendimento da poupança é de 3% ao ano, encontre uma fórmula fechada que expresse quanto essa pessoa terá na conta depois de um determinado número de anos. Por exemplo, 30 anos.

  O primeiro passo é definir o problema recursivamente:

  $T(0) = 100$ e

  $T(n) = T(n-1) + 0,03 \cdot T(n-1)$ para $n \geq 1$, ou seja,

  $T(n) = 1,03 \cdot T(n-1)$ para $n \geq 1$.

  Para, então, resolvê-la.

Exemplo

- **Expandir**:
  $k = 1$: $T(n) = 1,03T(n-1)$

Exemplo

- **Expandir**:
  $k = 1$: $T(n) = 1,03T(n-1)$
  $k = 2$: $T(n) = 1,03[1,03T(n-2)] = (1,03)^2T(n-2)$

Exemplo

- **Expandir**:
  $k = 1$: $T(n) = 1,03T(n-1)$
  $k = 2$: $T(n) = 1,03[1,03T(n-2)] = (1,03)^2T(n-2)$
  $k = 3$: $T(n) = (1,03)^2[1,03T(n-3)] = (1,03)^3T(n-3)$

Exemplo

- **Expandir**:
  $k = 1$: $T(n) = 1,03T(n-1)$
  $k = 2$: $T(n) = 1,03[1,03T(n-2)] = (1,03)^2T(n-2)$
  $k = 3$: $T(n) = (1,03)^2[1,03T(n-3)] = (1,03)^3T(n-3)$

- **Conjecturar**: Após k expansões, $T(n) = (1,03)^kT(n-k)$.

  Observando a expansão, ela irá parar quando $k = n$, isso porque a base da recursividade é definida para 0.

  Logo, $T(n) = (1,03)^n\ 100$.

Exemplo

- **Expandir**:
  $k = 1$: $T(n) = 1,03T(n-1)$
  $k = 2$: $T(n) = 1,03[1,03T(n-2)] = (1,03)^2T(n-2)$
  $k = 3$: $T(n) = (1,03)^2[1,03T(n-3)] = (1,03)^3T(n-3)$

- **Conjecturar**: Após k expansões, $T(n) = (1,03)^kT(n-k)$.

  Observando a expansão, ela irá parar quando $k = n$, isso porque a base da recursividade é definida para 0.

  Logo, $T(n) = (1,03)^n\ 100$.

- Depois de 30 anos, essa pessoa terá na conta:

  $T(30) = (1,03)^{30}\ 100$, ou seja, R$ 242,73.

Exercícios

1. Encontre a solução fechada para a sequência T definida por $T(1) = 1$ e $T(n) = T(n-1) + 3$ para $n \geq 2$.

2. Encontre a solução fechada para a sequência T definida por $T(1) = 7$ e $T(n) = 2T(n-1) + 1$ para $n \geq 2$.

3. Encontre a solução fechada para a sequência T definida por $T(1) = 1$ e $T(n) = T(\frac{n}{2}) + 1$ para $n \geq 2$.

4. Defina recursivamente cada uma das sequências abaixo.

   a) 1, 3, 5, 7, 9, ...
   b) 1, 4, 9, 16, 25, ...
   c) 1, 3, 7, 15, 31, 63.

Algoritmos recursivos

- Um algoritmo recursivo chama a si mesmo para decompor um problema maior em problemas menores e semelhantes.
- As soluções desses “subproblemas” são então combinadas para resolver o problema original.
- Eles seguem uma abordagem de **dividir para conquistar**:
  1. **Dividir** o problema em um ou mais subproblemas.
  2. **Conquistar** os subproblemas, resolvendo-os recursivamente. Se os tamanhos dos subproblemas forem pequenos o bastante, basta resolvê-los de forma direta.
  3. **Combinar** as soluções encontradas dos subproblemas, a fim de forma uma solução para o problema original.

Algoritmos recursivos

Para analisar a complexidade de um algoritmo recursivo, é preciso obter sua definição recursiva:

1. Determinar qual é o tamanho n do problema.
2. Verificar que valor n_0 é usado como base da recorrência.
3. Determinar $T(n_0)$ que geralmente é um número específico:

   $T(n_0) = c$, onde c é uma constante.
4. O passo recorrente $T(n)$ será igual a soma de a ocorrências de $T(f(n))$ mais a soma das outras instruções efetuadas $g(n)$:

   $T(n) = aT(f(n)) + g(n)$ para $n > n_0$.

Algoritmos recursivos

- **Exemplo 1**: O algoritmo abaixo calcula o fatorial de um número inteiro não-negativo *n* recursivamente.

```
Fatorial(n)
se (n = 0) retorne (1)
senão
    temp  = Fatorial(n - 1)
    valor = n * temp
    retorne (valor)
```

`4! = 4 * 3!`
`        3! = 3 * 2!`
`                2! = 2 * 1!`
`                        1! = 1 * 0!`
	O contexto de cada chamada é empilhado

`                                0! = 1`	Condição de parada

`                        1! = 1`
`                2! = 2`
`        3! = 6`
`4! = 24`
	O contexto de cada chamada é desempilhado

Algoritmos recursivos

- Outra forma de escrever o algoritmo:

```
Fatorial(n)
1. se (n = 0) retorne (1)
2. senão retorne (n * Fatorial(n - 1))
```

- Observe que a própria função é chamada **apenas uma vez** dentro do corpo da função.

- Para encontrar a complexidade no tempo do algoritmo fatorial recursivo, obtém-se sua definição recursiva:

  $T(0) = \Theta(1)$ e
  $T(n) = T(n-1) + \Theta(1)$ para $n > 0$.

- Expandindo $T(n)$, tem-se:

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

- Após k expansões a equação tem a forma:

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

- A expansão para quando $n-k=0$, ou seja, $k=n$. Logo,

$$T(n) = T(n-n)+n = T(0)+n = 1+n.$$

- Conclui-se que o algoritmo executa em tempo linear.

Algoritmos recursivos

- Outra maneira de analisar a complexidade no tempo é pensar na operação básica ou crucial do algoritmo.
- A operação básica do algoritmo Fatorial recursivo é o **número de multiplicações** indicado por $T(n)$.
- Assim, o número de multiplicações é definido por

  $T(n) = T(n-1) + 1$ para $n > 0$.

- Incluindo a condição inicial, ou seja, o caso base do algoritmo, chega-se a seguinte definição recursiva:

  $T(0) = 0$ e
  $T(n) = T(n-1) + 1$ para $n > 0$.

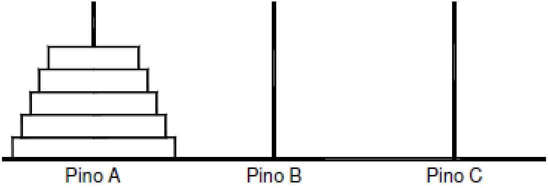
Algoritmos recursivos

- Já o algoritmo para calcular o fatorial de forma iterativa (não recursiva) pode ser facilmente implementado.

```
Fatorial(n)
1. x = 1
2. para i = 2 até n faça
3.     x = x * i
4. retorne (x)
```

- Os algoritmos recursivos são mais compactos, mais legíveis e mais fáceis de serem compreendidos.
- Em função das alocações de memória, os algoritmos recursivos tendem a ser mais lentos que os equivalentes iterativos.
- O Fatorial recursivo usa uma pilha, o que o torna mais "lento" que o iterativo, embora ambos sejam $O(n)$ no tempo.

- **Exemplo 2**: O algoritmo para resolver o quebra-cabeça conhecido como Torre de Hanói é recursivo.



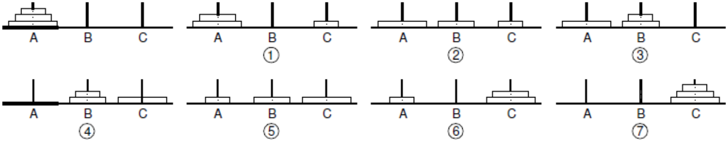
- Objetivo: Transferir todos os n discos de A (pino de origem) para C (pino de destino) utilizando o pino B como auxiliar.

- Regras: Mover um disco por vez e nunca colocar um disco maior em cima de um menor.

- Algoritmo:

```
Hanoi (n, A, C, B)
1. se (n = 1) então
2.     mover disco do topo de A para C
3. senão
4.     Hanoi(n-1, A, B, C)
5.     mover disco do topo de A para C
6.     Hanoi(n-1, B, C, A)
```

- Uma chamada com $n = 3$ gera uma saída com 7 movimentos:



Algoritmos recursivos

- Como visto, uma definição recursiva é dada por

  $T(n_0) = c$ (c é uma constante) e
  $T(n) = aT(f(n)) + g(n)$ para $n > n_0$.

- Por exemplo, para o algoritmo Fatorial recursivo, tem-se:

  $T(0) = 0$ e
  $T(n) = T(n-1) + 1$ para $n > 0$.

- Já para o algoritmo Hanói, onde a operação básica é o **número de movimentos**, tem-se:

  $T(1) = 1$ e
  $T(n) = T(n-1) + T(n-1) + 1 = 2T(n-1) + 1$ para $n > 1$.

- **Expandir**:
  $k = 1$: $T(n) = 2T(n-1) + 1$

Algoritmos recursivos

- **Expandir**:
  $k = 1$: $T(n) = 2T(n-1) + 1$
  $k = 2$: $T(n) = 2[2T(n-2) + 1] + 1 = 4T(n-2) + 3$

Algoritmos recursivos

- **Expandir**:
  $k = 1$: $T(n) = 2T(n-1) + 1$
  $k = 2$: $T(n) = 2[2T(n-2) + 1] + 1 = 4T(n-2) + 3$
  $k = 3$: $T(n) = 4[2T(n-3) + 1] + 3 = 8T(n-3) + 7$

- **Expandir**:
  $k = 1$: $T(n) = 2T(n-1) + 1$
  $k = 2$: $T(n) = 2[2T(n-2) + 1] + 1 = 4T(n-2) + 3$
  $k = 3$: $T(n) = 4[2T(n-3) + 1] + 3 = 8T(n-3) + 7$

- **Conjecturar**: Após k expansões, $T(n) = 2^k T(n-k) + 2^k - 1$.

  Observando a expansão, ela irá parar quando $k = n - 1$, isso porque a base da recursividade é definida para 1. Logo,

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

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

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

- **Expandir**:
  $k = 1$: $T(n) = 2T(n-1) + 1$
  $k = 2$: $T(n) = 2[2T(n-2) + 1] + 1 = 4T(n-2) + 3$
  $k = 3$: $T(n) = 4[2T(n-3) + 1] + 3 = 8T(n-3) + 7$

- **Conjecturar**: Após k expansões, $T(n) = 2^k T(n-k) + 2^k - 1$.

  Observando a expansão, ela irá parar quando $k = n - 1$, isso porque a base da recursividade é definida para 1. Logo,

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

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

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

- Para solucionar um Hanói de 64 discos, como diz a lenda, são necessários 18.446.744.073.709.551.615 movimentos.

Algoritmos recursivos

- Resolvendo a relação de recorrência, determina-se o tempo de execução (complexidade no tempo) do algoritmo recursivo.

- Por exemplo, o algoritmo recursivo da Torre de Hanói tem complexidade igual a $2^n - 1$, ou seja, $O(2^n)$.

- Um **erro** é analisar a complexidade do Fatorial recursivo através relação de recorrência: $T(n) = n \cdot T(n-1)$.

- É incorreto porque a própria função não é chamada n vezes dentro do corpo da função.

Exemplo 3

- A sequência de Fibonacci é $0, 1, 1, 2, 3, 5, 8, 13, 21, 34, \ldots$.

- Abaixo sua implementação recursiva:

```
FIB(n)
1. se (n = 0 ou n = 1)
2.     então retorne (n)
3.     senão retorne (FIB(n - 2) + FIB(n - 1))
```

- Sua definição recursiva é dada por

$$T(0) = 0, \ T(1) = 1 \text{ e}$$
$$T(n) = T(n-2) + T(n-1) \text{ para } n > 1.$$

Exemplo 3

- Limite inferior para $T(n)$:

$$T(n) \geq T(n-2) + T(n-2) = 2T(n-2)$$
$$T(n) \geq 4T(n-4)$$
$$T(n) \geq 8T(n-6)$$
$$\ldots$$
$$T(n) \geq 2^k T(n-2k)$$

- A expansão para quando $n - 2k = 1$, ou seja, $k = \frac{n-1}{2}$.

  Logo, $T(n) \geq 2^{\frac{n-1}{2}}$.

- Conclui-se que esse algoritmo recursivo é ineficiente, levando um tempo exponencial para calcular FIB(n).

Exemplo 3

- Para $n > 1$, cada chamada da função FIB causa duas novas chamadas de si mesma.
- Isso quer dizer que o número total de chamadas cresce exponencialmente.
- Por exemplo, para FIB(25) são feitas 242.784 chamadas recursivas!
- Alto custo com alocação de memória!

Exemplo 3

- Algoritmo iterativo para calcular a série de Fibonacci:

```
Fibonacci(n)
  se (n = 0 ou n = 1)
  então retorne (n)
  senão
    F1 = 0
    F2 = 1
    para i = 1 até n - 1 faça
       F = F1 + F2
       F1 = F2
       F2 = F
    retorne (F)
```

- Esse algoritmo tem complexidade no tempo $O(n)$ e não faz uso de uma pilha de execução.

- **Conclusão**: Não usar a recursividade cegamente!

Exemplo 4

- Algoritmo recursivo para pesquisa binária: Retorna a posição do elemento v no vetor A, caso ele lá se encontre.

```
BINARY-SEARCH(A,v,menor,maior)
1. se menor > maior então retorne NULO
2. se (menor = maior e v = A[menor])
3.    então retorne menor
4.    senão retorne NULO
5. meio = (menor + maior) / 2
6. se v = A[meio] então retorne meio
7. se v > A[meio]
8.    então retorne BINARY-SEARCH(A,v,meio+1,maior)
9.    senão retorne BINARY-SEARCH(A,v,menor,meio-1)
```

Exemplo 4

- A definição recursiva do algoritmo assume a fórmula abaixo, deduzindo que um vetor de entrada com 1 elemento tem complexidade constante.

  $T(1) = \Theta(1)$ e

  $T(n) = T(\frac{n}{2}) + \Theta(1)$ para $n > 1$.

- Para encontrar a complexidade no tempo do algoritmo basta resolver sua relação de recorrência.

Exemplo 4

- **Expandir**:
  $k = 1$: $T(n) = T(\frac{n}{2}) + 1$

Exemplo 4

- **Expandir**:
  $k = 1$: $T(n) = T(\frac{n}{2}) + 1$
  $k = 2$: $T(n) = [T(\frac{n}{4}) + 1] + 1 = T(\frac{n}{4}) + 2$

Exemplo 4

- **Expandir**:
  $k = 1$: $T(n) = T(\frac{n}{2}) + 1$
  $k = 2$: $T(n) = [T(\frac{n}{4}) + 1] + 1 = T(\frac{n}{4}) + 2$
  $k = 3$: $T(n) = [T(\frac{n}{8}) + 1] + 2 = T(\frac{n}{8}) + 3$

Exemplo 4

- **Expandir**:
  $k = 1$: $T(n) = T(\frac{n}{2}) + 1$
  $k = 2$: $T(n) = [T(\frac{n}{4}) + 1] + 1 = T(\frac{n}{4}) + 2$
  $k = 3$: $T(n) = [T(\frac{n}{8}) + 1] + 2 = T(\frac{n}{8}) + 3$

- **Conjecturar**:
  Após k expansões, temos $T(n) = T(\frac{n}{2^k}) + k$.

Exemplo 4

- **Expandir**:
  $k = 1$: $T(n) = T(\frac{n}{2}) + 1$
  $k = 2$: $T(n) = [T(\frac{n}{4}) + 1] + 1 = T(\frac{n}{4}) + 2$
  $k = 3$: $T(n) = [T(\frac{n}{8}) + 1] + 2 = T(\frac{n}{8}) + 3$

- **Conjecturar**:
  Após k expansões, temos $T(n) = T(\frac{n}{2^k}) + k$.

  A expansão irá parar em $n = 2^k$, ou seja, $k = log_2(n)$. Logo,

  $T(n) = T(\frac{n}{n}) + log_2(n) = 1 + log_2(n)$.

Exemplo 4

- **Expandir**:
  $k = 1$: $T(n) = T(\frac{n}{2}) + 1$
  $k = 2$: $T(n) = [T(\frac{n}{4}) + 1] + 1 = T(\frac{n}{4}) + 2$
  $k = 3$: $T(n) = [T(\frac{n}{8}) + 1] + 2 = T(\frac{n}{8}) + 3$

- **Conjecturar**:
  Após k expansões, temos $T(n) = T(\frac{n}{2^k}) + k$.

  A expansão irá parar em $n = 2^k$, ou seja, $k = log_2(n)$. Logo,

  $T(n) = T(\frac{n}{n}) + log_2(n) = 1 + log_2(n)$.

- A complexidade no tempo do algoritmo é $O(log_2(n))$.

Exemplo 4

- Para verificar a solução da fórmula fechada precisamos agora demonstrar sua validade por indução matemática em n.

Exemplo 4

- Para verificar a solução da fórmula fechada precisamos agora demonstrar sua validade por indução matemática em n.

  Hipótese Indutiva: $T(n) = 1 + log_2(n)$.

Exemplo 4

- Para verificar a solução da fórmula fechada precisamos agora demonstrar sua validade por indução matemática em n.

  Hipótese Indutiva: $T(n) = 1 + log_2(n)$.

  $T(1) = 1 + log_2(1) = 1$ OK!

Exemplo 4

- Para verificar a solução da fórmula fechada precisamos agora demonstrar sua validade por indução matemática em n.

  Hipótese Indutiva: $T(n) = 1 + log_2(n)$.

  $T(1) = 1 + log_2(1) = 1 \quad$ OK!

  Agora, queremos demonstrar que $T(n+1) = 1 + log_2(n+1)$.

Exemplo 4

- Para verificar a solução da fórmula fechada precisamos agora demonstrar sua validade por indução matemática em n.

  Hipótese Indutiva: $T(n) = 1 + log_2(n)$.

  $T(1) = 1 + log_2(1) = 1$ OK!

  Agora, queremos demonstrar que $T(n+1) = 1 + log_2(n+1)$.

  Da relação de recorrência e usando a hipótese indutiva:

Exemplo 4

- Para verificar a solução da fórmula fechada precisamos agora demonstrar sua validade por indução matemática em n.

  Hipótese Indutiva: $T(n) = 1 + log_2(n)$.

  $T(1) = 1 + log_2(1) = 1$ OK!

  Agora, queremos demonstrar que $T(n+1) = 1 + log_2(n+1)$.

  Da relação de recorrência e usando a hipótese indutiva:

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

Exemplo 4

- Para verificar a solução da fórmula fechada precisamos agora demonstrar sua validade por indução matemática em n.

  Hipótese Indutiva: $T(n) = 1 + log_2(n)$.

  $T(1) = 1 + log_2(1) = 1 \quad \text{OK!}$

  Agora, queremos demonstrar que $T(n+1) = 1 + log_2(n+1)$.

  Da relação de recorrência e usando a hipótese indutiva:

  $$T(n+1) = T(\frac{n+1}{2}) + 1$$
  $$T(n+1) = 1 + log_2(\frac{n+1}{2}) + 1$$

Exemplo 4

- Para verificar a solução da fórmula fechada precisamos agora demonstrar sua validade por indução matemática em n.

  Hipótese Indutiva: $T(n) = 1 + log_2(n)$.

  $T(1) = 1 + log_2(1) = 1$ OK!

  Agora, queremos demonstrar que $T(n+1) = 1 + log_2(n+1)$.

  Da relação de recorrência e usando a hipótese indutiva:

  $$T(n+1) = T(\tfrac{n+1}{2}) + 1$$
  $$T(n+1) = 1 + log_2(\tfrac{n+1}{2}) + 1$$
  $$T(n+1) = 1 + log_2(n+1) - log_2(2) + 1$$

Exemplo 4

- Para verificar a solução da fórmula fechada precisamos agora demonstrar sua validade por indução matemática em n.

  Hipótese Indutiva: $T(n) = 1 + log_2(n)$.

  $T(1) = 1 + log_2(1) = 1$ OK!

  Agora, queremos demonstrar que $T(n+1) = 1 + log_2(n+1)$.

  Da relação de recorrência e usando a hipótese indutiva:

  $$T(n+1) = T(\tfrac{n+1}{2}) + 1$$
  $$T(n+1) = 1 + log_2(\tfrac{n+1}{2}) + 1$$
  $$T(n+1) = 1 + log_2(n+1) - log_2(2) + 1$$
  $$T(n+1) = 1 + log_2(n+1) - 1 + 1$$

Exemplo 4

- Para verificar a solução da fórmula fechada precisamos agora demonstrar sua validade por indução matemática em n.

  Hipótese Indutiva: $T(n) = 1 + log_2(n)$.

  $T(1) = 1 + log_2(1) = 1$ OK!

  Agora, queremos demonstrar que $T(n+1) = 1 + log_2(n+1)$.

  Da relação de recorrência e usando a hipótese indutiva:

  $$T(n+1) = T(\tfrac{n+1}{2}) + 1$$
  $$T(n+1) = 1 + log_2(\tfrac{n+1}{2}) + 1$$
  $$T(n+1) = 1 + log_2(n+1) - log_2(2) + 1$$
  $$T(n+1) = 1 + log_2(n+1) - 1 + 1$$
  $$T(n+1) = 1 + log_2(n+1) \quad \text{c.q.d}$$

Teorema Mestre

- Sejam $a \geq 1$ e $b > 1$ constantes, $f(n)$ uma função e $T(n)$ definida sobre os inteiros não negativos pela recorrência

  $T(n) = aT(\frac{n}{b}) + f(n)$,

  para b uma potência de n. Então, $T(n)$ pode ser limitada assintoticamente como a seguir.

  1. Se $f(n) = O(n^{log_b a - \epsilon})$ para alguma constante $\epsilon > 0$, então $T(n) = \Theta(n^{log_b a})$.
  2. Se $f(n) = \Theta(n^{log_b a})$, então $T(n) = \Theta(n^{log_b a} log(n))$.
  3. Se $f(n) = \Omega(n^{log_b a + \epsilon})$ para alguma constante $\epsilon > 0$, e se $af(n/b) \leq c\ f(n)$ para alguma constante $c < 1$ e todo n suficientemente grande, então $T(n) = \Theta(f(n))$.

Método Mestre

- O problema a ser resolvido é dividido em a subproblemas de tamanho n/b cada um.
- Os a subproblemas são resolvidos recursivamente em tempo $T(n/b)$ cada um.
- A função $f(n)$ descreve o custo de dividir o problema em subproblemas e de combinar os resultados de cada um.
- Por exemplo, na recorrência $T(n) = T(\frac{n}{2}) + 1$, tem-se $a = 1$, $b = 2$ e $f(n) = 1$.
- Por exemplo, na recorrência $T(n) = 2T(\frac{n}{3}) + n - 1$, tem-se $a = 2$, $b = 3$ e $f(n) = n - 1$.

Método Mestre

- Resolva as seguintes relações de recorrência:

1. $T(n) = 4T(\frac{n}{2}) + n$
2. $T(n) = 2T(\frac{n}{2}) + n - 1$
3. $T(n) = T(\frac{2n}{3}) + n$
4. $T(n) = 3T(\frac{n}{3}) + nlog(n)$

Método Mestre

- Considere a relação de recorrência:

  $T(n) = 4T(\frac{n}{2}) + n,$

  onde $a = 4$, $b = 2$, $f(n) = n$ e $n^{log_b a} = n^{log_2 4} = n^2$. O caso 1 se aplica porque $f(n) = O(n^{log_b a - \epsilon}) = O(n)$, onde $\epsilon = 1$, e a solução é $T(n) = \Theta(n^2)$.

- Considere a relação de recorrência:

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

  onde $a = 2$, $b = 2$, $f(n) = n - 1$ e $n^{log_b a} = n^{log_2 2} = n$. O caso 2 se aplica porque $f(n) = \Theta(n^{log_b a}) = \Theta(n)$, e a solução é $T(n) = \Theta(nlog(n))$.

Método Mestre

- Considere a relação de recorrência:

  $T(n) = T(\frac{2n}{3}) + n,$

  onde $a = 1$, $b = 3/2$, $f(n) = n$ e $n^{log_b a} = n^{log_{3/2} 1} = n^0 = 1$. O caso 3 se aplica porque $f(n) = \Omega(n^{log_{3/2} 1+\epsilon}) = \Omega(n)$, onde $\epsilon = 1$ e $af(n/b) = 2n/3 \leq cf(n) = 2n/3$, para $c = 2/3$ e $n \geq 0$. Logo, a solução é $T(n) = \Theta(f(n)) = \Theta(n)$.

- Considere a relação de recorrência:

  $T(n) = 3T(\frac{n}{3}) + nlog(n),$

  onde $a = 3$, $b = 3$, $f(n) = nlog(n)$ e $n^{log_b a} = n^{log_3 3} = n$. Os casos 1 e 2 não se aplicam. O caso 3 também não se aplica porque $f(n) = nlog(n)$ não é $\Omega(nn^{\epsilon})$. Portanto, o Teorema Mestre não se aplica. Solução: Exercício 4.4-2.

Conclusões

- A recursão é uma técnica que define um problema em termos de uma ou mais versões menores desse mesmo problema.
- A complexidade do algoritmo recursivo depende do tamanho e da taxa de redução do problema a cada invocação.
- Um programa recursivo é mais "elegante" que a sua versão iterativa, porém, exige mais espaço de memória.
- Então, a recursividade vale a pena para algoritmos complexos, cuja implementação iterativa é difícil e normalmente requer o uso explícito de uma pilha.
  - Dividir p/ conquistar (Ex: busca binária, mergesort, quicksort)
  - Caminhamento em árvores (Ex: pesquisa, backtracking)

Exercício 5

- O algoritmo de ordenação MERGE-SORT recebe o vetor A e os índices do primeiro p e do último r elementos.

```
MERGE-SORT (A, p, r)
1. se (p < r) então
2.     q = (p + r) / 2
3.     MERGE-SORT (A, p, q)
4.     MERGE-SORT (A, q + 1, r)
5.     MERGE (A, p, q, r)
```

- Considerando que a função MERGE é $\Theta(n)$, ache a definição recursiva que expressa a complexidade no tempo do algoritmo.
- Qual é a complexidade no tempo do algoritmo?

Exercício 6

- Encontre a complexidade no tempo de um algoritmo com a seguinte definição recursiva:

  $T(1) = 0$ e

  $T(n) = T(n-1) + c$, onde c é uma constante e $n > 1$.

Exercício 7

- Considere o algoritmo TESTE que recebe como parâmetro de entrada um número inteiro não-negativo n.

```
TESTE (n)
1. se (n = 0) então retorne (8)
2. senão retorne (TESTE(n - 1) + 2)
```

- Qual é o valor retornado pelo algoritmo para $n = 8$?
- Qual é a complexidade no tempo do algoritmo?

Exercício 8

- Considere o seguinte pseudo-código:

```
Subrotina Rec( parâmetro n)
if n ≤ 1
        Stop
else
        Ative Processo X
        Rec(n/2)
```

- Seja $T(n)$ o número de ativações do processo X.
- Quantas vezes o processo X será ativado para uma entrada n, onde n é um número natural?

Exercício 9

- Dois algoritmos recursivos têm sua complexidade no tempo expressa pelas definições recursivas abaixo. Qual desses algoritmos é o mais eficiente assintoticamente?

- Algoritmo 1:

  $T(1) = 1$ e
  $T(n) = T(\frac{n}{3}) + n$ para $n > 1$.

- Algoritmo 2:

  $T(1) = 1$ e
  $T(n) = 3T(\frac{n}{4}) + n^2$ para $n > 1$.

Exercício 10

- Suponha que a operação crucial do algoritmo abaixo é o fato de inspecionar um elemento. Ele inspeciona os *n* elementos de um conjunto de inteiros e, de alguma forma, isso permite descartar $2/3$ dos elementos e então fazer uma chamada recursiva sobre os $n/3$ elementos restantes.

```
        void pesquisa(n) {
(1)       if (n <= 1)
(2)         'inspecione elemento' e termine
          else {
(3)         para cada um dos n elementos 'inspecione elemento';
(4)         pesquisa(n/3);
          }
        }
```

- Apresente a complexidade no tempo do algoritmo.

Exercício 11

- Observe a função recursiva a seguir.

```
function Prova (N : integer) : integer;
begin
  if N = 0 then Prova :=  0
  else Prova := N * 2 - 1 + Prova (N - 1);
end;
```

- Considerando-se que essa função sempre será chamada com N contendo inteiros positivos, o seu valor de retorno será

  (a) O fatorial do valor armazenado em N.
  (b) O valor armazenado em N elevado ao quadrado.
  (c) O somatório dos N primeiros números inteiros positivos.
  (d) O somatório dos N primeiros números pares positivos.
  (e) 2 elevado ao valor armazenado em N.

Exercício 12

- A função recursiva abaixo calcula a potência de um número.

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

- Qual é a complexidade no tempo da função?