

Árvores e processamento simultâneo

Marco A L Barbosa
malbarbo.pro.br

Departamento de Informática
Universidade Estadual de Maringá



Este trabalho está licenciado com uma Licença Creative Commons - Atribuição-Compartilhável 4.0 Internacional.

<http://github.com/malbarbo/na-progfun>

Árvores binárias

Como podemos definir uma árvore binária?



Uma `ÁrvoreBinária` é

- `empty`; ou
- `(no Número ÁrvoreBinária ÁrvoreBinária)`, onde `no` é uma estrutura com os campos `valor`, `esq` e `dir`

```
(struct no (valor esq dir) #:transparent)
```

Modelo

```
(define (fn-para-ab t)
  (cond
    [(empty? t) ...]
    [else
     (... (no-valor t)
          (fn-para-ab (no-esq t))
          (fn-para-ab (no-dir t)))]))
```

Defina uma função que calcule a altura de uma árvore binária. A altura de uma árvore binária é a distância entre a raiz e o seu descendente mais afastado. Uma árvore com um único nó tem altura 0.

Exemplo: altura árvore

```
;;      t4  3
;;      /  \
;; t3  4    7  t2
;;      /    / \
;;      3    8  9  t1
;;              /
;;             t0 10

(define t0 (no 10 empty empty))
(define t1 (no 9 t0 empty))
(define t2 (no 7 (no 8 empty empty) t1))
(define t3 (no 4 (no 3 empty empty) empty))
(define t4 (no 3 t3 t2))
```

```
;; ÁrvoreBinária -> Natural
;; Devolve a altura da árvore binária. A altura de
;; uma árvore binária é a distância da raiz a seu
;; descendente mais afastado. Uma árvore com um
;; único nó tem altura 0.
```

```
(examples
  (check-equal? (altura empty) ?)
  (check-equal? (altura t0) 0)
  (check-equal? (altura t1) 1)
  (check-equal? (altura t2) 2)
  (check-equal? (altura t3) 1)
  (check-equal? (altura t4) 3))

(define (altura t)
  (cond
    [(empty? t) ...]
    [else (... (no-valor t)
               (altura (no-esq t))
               (altura (no-dir t))))]))
```

Exemplo: altura árvore

```
;;      t4  3
;;      /  \
;;  t3  4    7  t2
;;      /    / \
;;      3    8  9  t1
;;           /
;;         t0  10
```

```
(define t0 (no 10 empty empty))
(define t1 (no 9 t0 empty))
(define t2 (no 7 (no 8 empty empty) t1))
(define t3 (no 4 (no 3 empty empty) empty))
(define t4 (no 3 t2 t3))
```

```
;; ÁrvoreBinária -> Natural
;; Devolve a altura da árvore binária. A altura de
;; uma árvore binária é a distância da raiz a seu
;; descendente mais afastado. Uma árvore com um
;; único nó tem altura 0. Uma árvore vazia tem
;; altura -1.
```

(examples

```
(check-equal? (altura empty) -1)
(check-equal? (altura t0) 0)
(check-equal? (altura t1) 1)
(check-equal? (altura t2) 2)
(check-equal? (altura t3) 1)
(check-equal? (altura t4) 3))
```

(define (altura t)

(cond

[(empty? t) -1]

[else (add1 (max (altura (no-esq t))

(altura (no-dir t))))))])

Listas aninhadas

Às vezes é necessário criar uma lista, que contenha outras listas, e estas listas contenham outras listas, etc.

```
> (list 1 4 (list 5 empty (list 2) 9) 10)
'(1 4 (5 () (2) 9) 10)
```

Chamamos este tipo de lista de lista aninhada. Como podemos definir uma lista aninhada?

Uma ListaAninhada é

- empty; ou
- (cons ListaAninhada ListaAninhada)
- (cons Número ListaAninhada)

Modelo

```
(define (fn-para-ladn ldn)
  (cond
    [(empty? ldn) ...]
    [(list? (first ldn))
     (... (fn-para-ladn (first ldn))
          (fn-para-ladn (rest ldn)))]
    [else
     (... (first ldn)
          (fn-para-ladn (rest ldn)))]))
```

Defina uma função que some todos os números de uma lista aninhada de números.

Exemplo: soma*

```
;; ListaAninhada -> Número
;; Devolve a soma de todos os elementos de ldn.
(examples
 (check-equal? (soma* empty)
               0)
 (check-equal? (soma* (list (list 1 (list empty 3)) (list 4 5) 4 6 7))
               30))
(define (soma* ldn)
  (cond
   [(empty? ldn) ...]
   [(list? (first ldn))
    (... (soma* (first ldn))
         (soma* (rest ldn)))]
   [else
    (... (first ldn)
         (soma* (rest ldn)))]))
```

Exemplo: soma*

```
;; ListaAninhada -> Número
;; Devolve a soma de todos os elementos de ldn.
(examples
  (check-equal? (soma* empty)
                0)
  (check-equal? (soma* (list (list 1 (list empty 3)) (list 4 5) 4 6 7))
                30))
(define (soma* ldn)
  (cond
    [(empty? ldn) 0]
    [(list? (first ldn))
     (+ (soma* (first ldn))
        (soma* (rest ldn)))]
    [else
     (+ (first ldn)
        (soma* (rest ldn)))]))
```

Defina uma função que aplaine uma lista aninhada, isto é, transforme uma lista aninhada em uma lista sem listas aninhadas com os mesmos elementos e na mesma ordem da lista aninhada.

Exemplo: aplaina

```
;; ListaAninhada -> ListaDeNúmeros
;; Devolve uma versão não aninhada de ldn, isto é, uma lista com os mesmos
;; elementos de ldn, mas sem aninhamento.
(examples
 (check-equal? (aplaina empty) empty)
 (check-equal? (aplaina (list (list 1 (list empty 3)) (list 4 5) 4 6 7))
                (list 1 3 4 5 4 6 7)))
(define (aplaina ldn)
  (cond
   [(empty? ldn) ...]
   [(list? (first ldn))
    (... (aplaina (first ldn))
         (aplaina (rest ldn)))]
   [else
    (... (first ldn)
         (aplaina (rest ldn)))]))
```

Exemplo: aplaina

```
;; ListaAninhada -> ListaDeNúmeros
;; Devolve uma versão não aninhada de ldn, isto é, uma lista com os mesmos
;; elementos de ldn, mas sem aninhamento.
(examples
 (check-equal? (aplaina empty) empty)
 (check-equal? (aplaina (list (list 1 (list empty 3)) (list 4 5) 4 6 7))
                (list 1 3 4 5 4 6 7)))
(define (aplaina ldn)
  (cond
   [(empty? ldn) empty]
   [(list? (first ldn))
    (append (aplaina (first ldn))
            (aplaina (rest ldn)))]
   [else
    (cons (first ldn)
          (aplaina (rest ldn)))]))
```

Processamento simultâneo

Como implementar uma função que consome dois argumentos e os dois são de tipos com autorreferência? Temos três possibilidades:

- 1) Tratar um dos argumentos como atômico e utilizar o modelo do tipo de dado do outro argumento.
- 2) Processar os dois argumentos de forma sincronizada.
- 3) Considerar todos os casos possíveis.

Projete uma função que concatene duas listas de números.

Exemplo: concatenação

```
;; ListaDeNúmeros ListaDeNúmeros -> ListaDeNúmeros
;; Produz uma nova lista com os elementos de lsta seguidos
;; dos elementos de lstb.
(examples
 (check-equal? (concatena empty
                        (cons 10 (cons 4 (cons 6 empty))))
               (cons 10 (cons 4 (cons 6 empty))))
 (check-equal? (concatena (cons 3 empty)
                        (cons 10 (cons 4 (cons 6 empty))))
               (cons 3 (cons 10 (cons 4 (cons 6 empty))))
 (check-equal? (concatena (cons 7 (cons 3 empty))
                        (cons 10 (cons 4 (cons 6 empty))))
               (cons 7 (cons 3 (cons 10 (cons 4 (cons 6 empty))))))
(define (concatena lsta lstb) empty)
```

Pelo propósito e pelos exemplos, qual dos argumentos pode ser tratado como atômico, isto é, não precisa ser decomposto? `lstb`.

Então usamos o modelo para processar `lsta`.

Exemplo: concatenação

```
;; ListaDeNúmero ListaDeNúmeros -> ListaDeNúmeros
;; Produz uma nova lista com os elementos de lsta seguidos
;; dos elementos de lstb.
(examples
 (check-equal? (concatena empty
                        (cons 10 (cons 4 (cons 6 empty))))
               (cons 10 (cons 4 (cons 6 empty))))
 (check-equal? (concatena (cons 3 empty)
                        (cons 10 (cons 4 (cons 6 empty))))
               (cons 3 (cons 10 (cons 4 (cons 6 empty))))
 (check-equal? (concatena (cons 7 (cons 3 empty))
                        (cons 10 (cons 4 (cons 6 empty))))
               (cons 7 (cons 3 (cons 10 (cons 4 (cons 6 empty))))))
(define (concatena lsta lstb)
  (cond
   [(empty? lsta) ... lstb]
   [else
    ... (first lsta)
        (concatena (rest lsta) lstb)]))
```

Exemplo: concatenação

```
;; ListaDeNúmero ListaDeNúmeros -> ListaDeNúmeros
;; Produz uma nova lista com os elementos de lsta seguidos
;; dos elementos de lstb.
(examples
 (check-equal? (concatena empty
                        (cons 10 (cons 4 (cons 6 empty))))
              (cons 10 (cons 4 (cons 6 empty))))
 (check-equal? (concatena (cons 3 empty)
                          (cons 10 (cons 4 (cons 6 empty))))
              (cons 3 (cons 10 (cons 4 (cons 6 empty))))
 (check-equal? (concatena (cons 7 (cons 3 empty))
                          (cons 10 (cons 4 (cons 6 empty))))
              (cons 7 (cons 3 (cons 10 (cons 4 (cons 6 empty))))))
(define (concatena lsta lstb)
  (cond
   [(empty? lsta) lstb]
   [else
    (cons (first lsta)
          (concatena (rest lsta) lstb))]))
```

Projete uma função que calcule a soma ponderada a partir de uma lista de números e uma lista de pesos.

Exemplo: soma ponderada

```
;; ListaDeNúmeros ListaDeNúmeros -> Número
;; Calcula a soma ponderada dos valores de lst considerando que cada
;; elemento de lst tem como peso o elemento correspondente em pesos.
;; Requer que lst e pesos tenham o mesmo tamanho
```

(examples

```
(check-equal? (soma-ponderada empty empty) 0)
(check-equal? (soma-ponderada (list 4) (list 2)) 8) ; (+ 0 (* 4 2))
(check-equal? (soma-ponderada (list 3 4) (list 5 2)) 23) ; (+ (* 3 5) (* 4 2))
(check-equal? (soma-ponderada (list 5 3 4) (list 1 5 2)) 28)) ; (+ (* 5 1) (* 3 5) (* 4 2))
```

(define (soma-ponderada lst pesos) 0)

O requisito de que `lst` e `pesos` têm o mesmo tamanho pode ser explorado no corpo inicial:

- Quando `lst` é vazia, `pesos` também é.
- Quando `lst` e `pesos` não são vazias, temos `(first lst)`, `(rest lst)`, `(first pesos)` e `(rest pesos)`
- Para a chamada recursiva, temos `(rest lst)` e `(rest pesos)`, que têm o mesmo tamanho.

Exemplo: soma ponderada

```
;; ListaDeNúmeros ListaDeNúmeros -> Número
;; Calcula a soma ponderada dos valores de lst cosiderando que cada
;; elemento de lst tem como peso o elemento correspondente em pesos.
;; Requer que lst e pesos tenham o mesmo tamanho
```

(examples

```
(check-equal? (soma-ponderada empty empty) 0)
(check-equal? (soma-ponderada (list 4) (list 2)) 8) ; (+ 0 (* 4 2))
(check-equal? (soma-ponderada (list 3 4) (list 5 2)) 23) ; (+ (* 3 5) (* 4 2))
(check-equal? (soma-ponderada (list 5 3 4) (list 1 5 2)) 28)) ; (+ (* 5 1) (* 3 5) (* 4 2))
```

(define (soma-ponderada lst pesos)

```
(cond
 [(empty? lst) ...]
 [else
  ... (first lst)
      (first pesos)
      (soma-ponderada (rest lst) (rest pesos))]))
```

Exemplo: soma ponderada

```
;; ListaDeNúmeros ListaDeNúmeros -> Número  
;; Calcula a soma ponderada dos valores de lst cosiderando que cada  
;; elemento de lst tem como peso o elemento correspondente em pesos.  
;; Requer que lst e pesos tenham o mesmo tamanho
```

(examples

```
(check-equal? (soma-ponderada empty empty) 0)  
(check-equal? (soma-ponderada (list 4) (list 2)) 8) ; (+ 0 (* 4 2))  
(check-equal? (soma-ponderada (list 3 4) (list 5 2)) 23) ; (+ (* 3 5) (* 4 2))  
(check-equal? (soma-ponderada (list 5 3 4) (list 1 5 2)) 28)) ; (+ (* 5 1) (* 3 5) (* 4 2))
```

(define (soma-ponderada lst pesos)

```
(cond  
  [(empty? lst) 0]  
  [else  
   (+ (* (first lst)  
         (first pesos))  
       (soma-ponderada (rest lst) (rest pesos)))]))
```

Dados duas listas `lsta` e `lstb`, defina uma função que verifique se `lsta` é prefixo de `lstb`, isto é `lstb` começa com `lsta`.

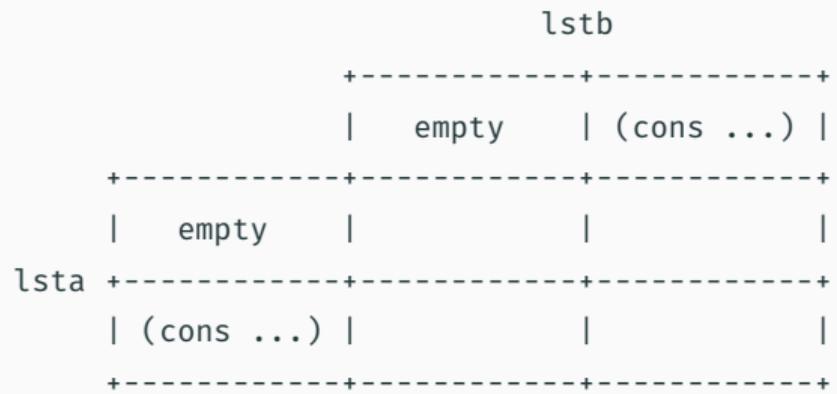
Especificação

```
;; Lista Lista -> Boolean  
;; Devolve #t se lsta é prefixo de lstb, #f caso contrário.  
(define (prefixo? lsta lstb) #f)
```

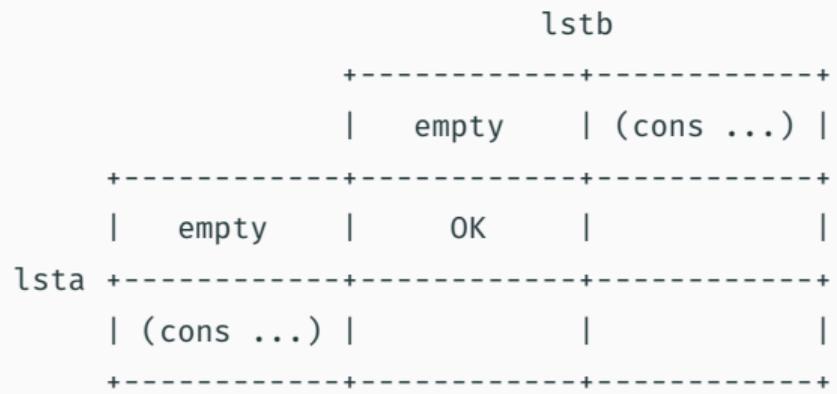
Exemplos

- Temos que ter pelo menos um exemplo para cada combinação das definições dos dados de entrada
- `lsta` pode ser `empty` ou um `cons`
- `lstb` pode ser `empty` ou um `cons`
- Como garantir que não vamos esquecer nenhum caso? Fazendo uma tabela!

Exemplo: prefixo



Exemplo: prefixo



```
(check-equal? (prefixo? empty empty) #t)
```

Exemplo: prefixo

```

                                lstb
                                +-----+
                                | empty  | (cons ...) |
+-----+-----+-----+
| empty  | OK   | OK   |
lstb +-----+-----+-----+
| (cons ...) |      |      |
+-----+-----+-----+
```

```
(check-equal? (prefixo? empty empty) #t)
```

```
(check-equal? (prefixo? empty (list 3 2 1)) #t)
```

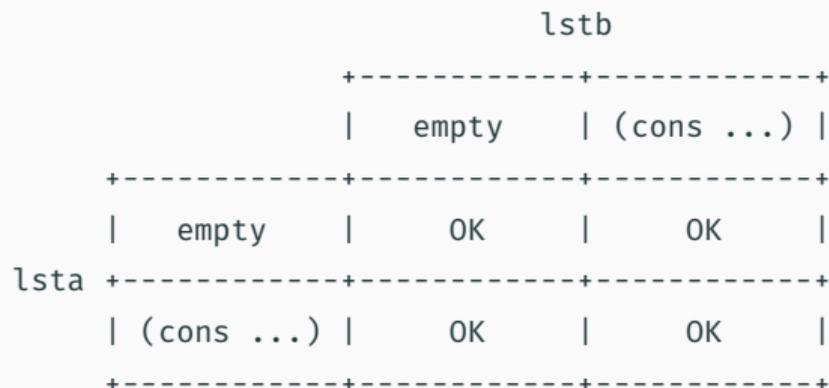
Exemplo: prefixo

```

                                lstb
                                +-----+-----+
                                | empty   | (cons ...) |
+-----+-----+-----+
| empty | OK   | OK   |
lstb +-----+-----+-----+
| (cons ...) | OK   |      |
+-----+-----+-----+
```

```
(check-equal? (prefixo? empty empty) #t)
(check-equal? (prefixo? empty (list 3 2 1)) #t)
(check-equal? (prefixo? (list 3 2 1) empty) #f)
```

Exemplo: prefixo



```
(check-equal? (prefixo? empty empty) #t)
(check-equal? (prefixo? empty (list 3 2 1)) #t)
(check-equal? (prefixo? (list 3 2 1) empty) #f)
(check-equal? (prefixo? (list 3 4) (list 3 4)) #t)
(check-equal? (prefixo? (list 3 4) (list 3 5)) #f)
(check-equal? (prefixo? (list 3 4) (list 3 4 6 8)) #t)
(check-equal? (prefixo? (list 3 5) (list 3 4 6 8)) #f)
(check-equal? (prefixo? (list 3 4 5) (list 3 4)) #f)
```

Implementação

Vamos começar criando um modelo com as quatro possibilidades

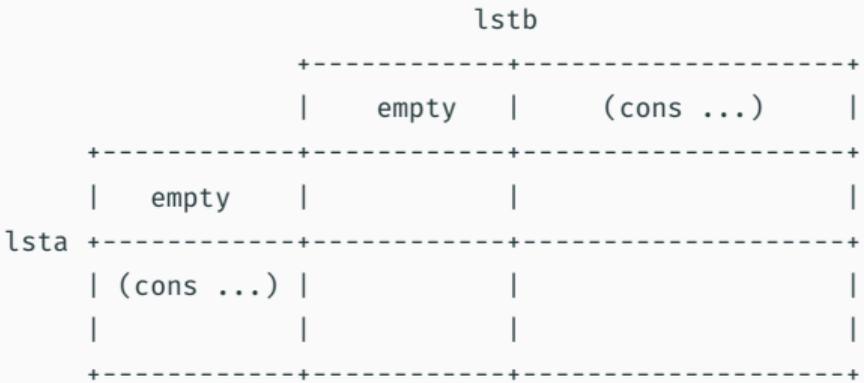
```
(define (prefixo? lsta lstb)
  (cond
    [(and (empty? lsta) (empty? lstb)) ...]
    [(and (empty? lsta) (cons? lstb)) ... lstb ...]
    [(and (cons? lsta) (empty? lstb)) ... lsta ...]
    [else ... lsta ... lstb ...]))
```

Este início é muito complicado...

Baseado nos exemplos, vamos preencher a tabela e derivar um código mais simples

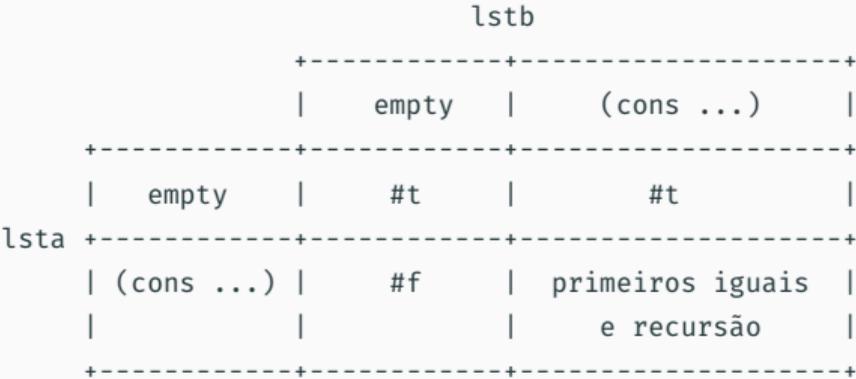
Exemplo: prefixo

```
(check-equal? (prefixo? empty empty) #t)
(check-equal? (prefixo? empty (list 3 2 1)) #t)
(check-equal? (prefixo? (list 3 2 1) empty) #f)
(check-equal? (prefixo? (list 3 4) (list 3 4)) #t)
(check-equal? (prefixo? (list 3 4) (list 3 5)) #f)
(check-equal? (prefixo? (list 3 4) (list 3 4 6 8)) #t)
(check-equal? (prefixo? (list 3 5) (list 3 4 6 8)) #f)
(check-equal? (prefixo? (list 3 4 5) (list 3 4)) #f)
```



Exemplo: prefixo

```
(check-equal? (prefixo? empty empty) #t)
(check-equal? (prefixo? empty (list 3 2 1)) #t)
(check-equal? (prefixo? (list 3 2 1) empty) #f)
(check-equal? (prefixo? (list 3 4) (list 3 4)) #t)
(check-equal? (prefixo? (list 3 4) (list 3 5)) #f)
(check-equal? (prefixo? (list 3 4) (list 3 4 6 8)) #t)
(check-equal? (prefixo? (list 3 5) (list 3 4 6 8)) #f)
(check-equal? (prefixo? (list 3 4 5) (list 3 4)) #f)
```



Exemplo: prefixo

Completando a implementação ...

```

                                lstb
                                +-----+-----+
                                |  empty  | (cons ...) |
+-----+-----+-----+-----+
|  empty  |                               | #t      |
lsta +-----+-----+-----+-----+
| (cons ...) | #f      | primeiros iguais |
|           |           | e recursão natural |
+-----+-----+-----+-----+
```

```
(define (prefixo? lsta lstb)
  (cond
    [(empty? lsta) #t]      ;; os casos foram
    [(empty? lstb) #f]     ;; escolhidos por ordem
    [else                   ;; de simplicidade
     (and (equal? (first lsta)
                  (first lstb))
          (prefixo? (rest lsta) (rest lstb))))]))
```

Defina uma função que encontre o k -ésimo elemento de uma lista.

Especificação

```
;; ListaDeNúmeros Natural -> Número  
;; Devolve o elemento na posição k da lst.  
;; O primeiro elemento está na posição 0.  
(define (lista-ref lst k) 0)
```


Exemplo: k -ésimo

Implementação

```
;; ListaDeNúmeros Natural -> Número
;; Devolve o elemento na posição k da lst.
;; O primeiro elemento está na posição 0.
;;
;;                               k
;;          +-----+-----+
;;          |      0      | (add1 ...) |
;;  +-----+-----+-----+
;;  |  empty  |          erro          |
;;  lst +-----+-----+-----+
;;  | (cons ...) | (first lst) |   recursão   |
;;  +-----+-----+-----+
(check-exn exn:fail? (thunk (lista-ref empty 0)))
(check-exn exn:fail? (thunk (lista-ref empty 2)))
(check-equal? (lista-ref (list 3 2 8) 0) 3)
(check-equal? (lista-ref (list 3 2 8 10) 2) 8)
(check-exn exn:fail? (thunk (lista-ref (list 3 2 8 10) 4))))
(define (lista-ref k lst) 0)
```

Exemplo: k -ésimo

```
;; ListaDeNúmeros Natural -> Número
;;
;;
;;          +-----+-----+
;;          |      0      | (add1 ...) |
;;  +-----+-----+
;;  | empty |      erro      |
;; lst +-----+-----+
;;  | (cons ...) | (first lst) |      recursão      |
;;  +-----+-----+
(check-exn exn:fail? (thunk (lista-ref empty 0)))
(check-exn exn:fail? (thunk (lista-ref empty 2)))
(check-equal? (lista-ref (list 3 2 8) 0) 3)
(check-equal? (lista-ref (list 3 2 8 10) 2) 8)
(check-exn exn:fail? (thunk (lista-ref (list 3 2 8 10) 4))))
(define (lista-ref k lst)
  (cond
    [(empty? lst) (error "Lista vazia")]
    [(zero? k) (first lst)]
    [else (lista-ref (rest lst) (sub1 k))]))
```

Básicas

- Capítulo 23 HTDP
- Vídeos 2 one-of