Custo de Computação

Fundamentos de Algoritmos

INF05008

Custo de uma Computação

- Tempo?
- Espaço?

Concreto $\times$ Abstrato?

Exemplo 1

```
(define (quantos? lista)
  (cond
    [(empty? lista) 0]
    [else (+ (quantos? (rest lista)) 1)]))

  (quantos? (list 'a 'b 'c))
= (+ (quantos? (list 'b 'c)) 1)
= (+ (+ (quantos? (list 'c)) 1) 1)
= (+ (+ (+ (quantos? empty) 1) 1) 1) = 3
```

Exemplo 1

```
(define (quantos? lista)
  (cond
    [(empty? lista) 0]
    [else (+ (quantos? (rest lista)) 1)]))

  (quantos? (list 'a 'b 'c))
= (+ (quantos? (list 'b 'c)) 1)
= (+ (+ (quantos? (list 'c)) 1) 1)
= (+ (+ (+ (quantos? empty) 1) 1) 1) = 3
```

$\Longrightarrow$ Quanto mais **longa** a entrada, mais **passos de recursão** são necessários

Exemplo 1

```
(define (quantos? lista)
  (cond
    [(empty? lista) 0]
    [else (+ (quantos? (rest lista)) 1)]))

  (quantos? (list 'a 'b 'c))
= (+ (quantos? (list 'b 'c)) 1)
= (+ (+ (quantos? (list 'c)) 1) 1)
= (+ (+ (+ (quantos? empty) 1) 1) 1) = 3
```

$\Longrightarrow$ Quanto mais **longa** a entrada, mais **passos de recursão** são necessários

$\Longrightarrow$ O **número de passos** entre as chamadas recursivas é **sempre o mesmo**

Função de Custo (Abstrato)

$\Longrightarrow$ Escolher **unidade de medida** : chamadas recursivas;

$\Longrightarrow$ Usar **tamanho da entrada** como variável

Função de Custo (Abstrato)

Qual computação custará mais:

```
(quantos?  (list 'a 'd 'b)),
(quantos?  (list 'a 'b 'c)), ou
(quantos?  (list 'b 'b 'b))?
```

Exemplo 2

```
(define (contém-b? lds)
  (cond
    [(empty? lds) false]
    [else
      (cond
        [(symbol=? (first lds) 'b) true]
        [else (contém-b? (rest lds))])]))
```

Exemplo 2

```
(define (contém-b? lds)
  (cond
    [(empty? lds) false]
    [else
      (cond
        [(symbol=? (first lds) 'b) true]
        [else (contém-b? (rest lds))])]))
```

$\Longrightarrow$ O número de passos necessários varia conforme a estrutura da entrada.

Considerar melhor caso, pior caso ou caso médio?

Exemplo 2

```
(define (contém-b? lds)
  (cond
    [(empty? lds) false]
    [else
      (cond
        [(symbol=? (first lds) 'b) true]
        [else (contém-b? (rest lds))])]))
```

$\Longrightarrow$ O número de passos necessários varia conforme a estrutura da entrada. **Considerar melhor caso, pior caso ou caso médio?**

$\Longrightarrow$ O tempo de cada recursão é **abstrato**. Portanto, podemos ignorar constantes e usar **Ordens de Grandeza**

Exemplo 2

```
(define (contém-b? lds)
  (cond
    [(empty? lds) false]
    [else
      (cond
        [(symbol=? (first lds) 'b) true]
        [else (contém-b? (rest lds))])]))
```

$\Longrightarrow$ O número de passos necessários varia conforme a estrutura da entrada. **Considerar melhor caso, pior caso ou caso médio?**

$\Longrightarrow$ O tempo de cada recursão é **abstrato**. Portanto, podemos ignorar constantes e usar **Ordens de Grandeza**

$\Longrightarrow$ O número de passos (chamadas recursivas) para esta função chegar ao resultado é, em média, $N/2$, sendo N o tamanho da entrada. Portanto, este algoritmo é da **ordem de N passos, ou $O(N)$** .

Função de Custo usando $O(N)$

Qual computação custará mais?

```
(quantos?  (list 'a 'd 'b)),
(quantos?  (list 'a 'b 'c)), ou
(quantos?  (list 'b 'b 'b))?
```

Qual é o valor de N em cada caso?

Exemplo 3

```
(define (ordena ldn)
  (cond
    [(empty? ldn) empty]
    [(cons? ldn) (insere (first ldn) (ordena (rest ldn)))]))

(define (insere n ldn)
  (cond
    [(empty? ldn) (cons n empty)]
    [else (cond
        [(>= n (first ldn)) (cons n ldn)]
        [(<  n (first ldn)) (cons (first ldn)
                                  (insere n (rest ldn)))])]))
```

$\Longrightarrow$ Quantos passos (chamadas recursivas) esta função executa em média para chegar ao resultado?

Exemplo de execução de sort

```
   (ordena (list 3 1 2))
= (insere 3 (ordena (list 1 2)))
= (insere 3 (insere 1 (ordena (list 2))))
= (insere 3 (insere 1 (insere 2 (ordena empty))))
= (insere 3 (insere 1 (insere 2 empty)))
= (insere 3 (insere 1 (list 2)))
= (insere 3 (cons 2 (insere 1 empty)))
= (insere 3 (cons 2 (list 1)))
= (insere 3 (list 2 1)) = (list 3 2 1)
```

Exemplo 3

```
(define (ordena ldn)
  (cond
    [(empty? ldn) empty]
    [(cons? ldn) (insere (first ldn) (ordena (rest ldn)))]))

(define (insere n ldn)
  (cond
    [(empty? ldn) (cons n empty)]
    [else (cond
        [(>= n (first ldn)) (cons n ldn)]
        [(<  n (first ldn)) (cons (first ldn)
                                  (insere n (rest ldn)))])]))
```

$\Longrightarrow$ Sendo N o tamanho da lista de entrada, temos, em média:

- $O(N)$ chamadas de `ordena`
- $O(N^2)$ chamadas de `insere`

Exemplo 4

```
;; maior : lista-de-números -> número
;; Determina o maior de uma lista de números
(define (maior ldn)
  (cond
    [(empty? (rest ldn)) (first ldn)]
    [else
      (cond
        [(> (maior (rest ldn)) (first ldn)) (maior (rest ldn))]
        [else (first ldn)])]))
```

Exemplo 4

```
;; maior : lista-de-números -> número
;; Determina o maior de uma lista de números
(define (maior ldn)
  (cond
    [(empty? (rest ldn)) (first ldn)]
    [else
      (cond
        [(> (maior (rest ldn)) (first ldn)) (maior (rest ldn))]
        [else (first ldn)])]))
```

Expressão	Requer 2 avaliações de
`(maior (list 0 1 2 3))`	`(maior (list 1 2 3))`
`(maior (list 1 2 3))`	`(maior (list 2 3))`
`(maior (list 2 3))`	`(maior (list 3))`

Exemplo 4'

```
;; maior2 : lista-de-números -> número
;; Determina o maior de uma lista de números
(define (maior2 ldn)
  (cond
    [(empty? (rest ldn)) (first ldn)]
    [else (local (
            (define maior-do-resto (maior2 (rest ldn))) )
            (cond
               [(> maior-do-resto (first ldn)) maior-do-resto]
               [else (first ldn)])])))}
```

Ordens de Grandeza

O **custo de um programa** , com relação ao **tamanho da entrada**, pode ser de ordem:

- Constante: $O(1)$
- Linear: $O(N)$
- Logarítmica: $O(log(N))$
- Fatorial: $O(!N)$
- Polinomial: $O(N^C)$, onde $C \in \mathbb{R}$
- Exponencial: $O(C^N)$, onde $C \in \mathbb{R}$ e $C > 1$

Ordens de Grandeza

- Usando esse custo, pode-se dividir os problemas computáveis em **classes**
- A partir da classe do problema, podemos saber se existe ou não um **algoritmo eficiente** que possa resolvê-lo
- Um algoritmo eficiente é aquele que resolve o problema dentro de um **tempo razoável**
- Um algoritmo $O(1)$ para resolver um determinado problema é chamado de **algoritmo ótimo**, porém encontrar tal algoritmo é geralmente improvável, ou até impossível...

Em outras disciplinas...

- Estruturas de dados (INA-2)
- Lógica para computação (INT-2)
- Teoria do grafos e análise combinatória (INT-2)
- Classificação e pesquisa de dados (INA-3)
- Computação simbólica e numérica (INT-3)
- Teoria da computação (INT-3)
- Linguagens formais e autômatos (INT-4)
- Complexidade de algoritmos (INT-5)
- Semântica formal (INT-5)