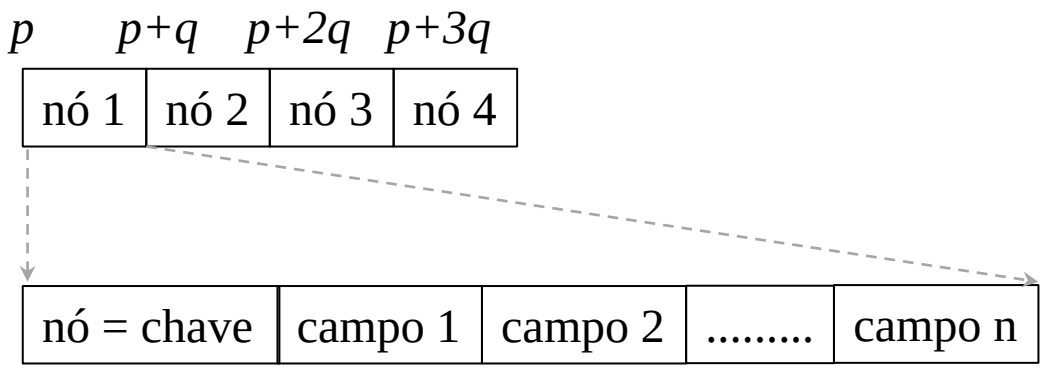
Algoritmo de busca sequencial

profº Mauricio Conceição Mario

Listas lineares em alocação sequencial

→ Uma ***lista linear*** é caracterizada pelo fato de seus nós estarem em posições contíguas, ou seja, o endereço do (*j + 1*)-*ésimo* nó da lista se encontra ***q*** unidades adiante do nó correspondente ao *j-ésimo*. A constante ***q*** é o número de palavras de memória que cada nó ocupa. Exemplo:

p = posição de memória



(Szwarcfiter & Markenzon, 2020)

Busca em listas lineares em alocação sequencial → manipulação:

→ Seja uma ***lista linear L*** com *n* elementos. Implementar algoritmo que busque um nó nesta lista, partindo da premissa que se tem conhecimento da identificação de suas chaves (nós):

```
L = [chave 1, campo 1, campo 2, ...,campo k, chave 2, ...]
n = dimensão (L)
        função busca_chave(nó)
            i := 1
            enquanto (i < n)  fazer:
                se ( L[i] == nó ) fazer:
                    retorna i
                senão
                    i := i + 1
busca_chave(x)
```

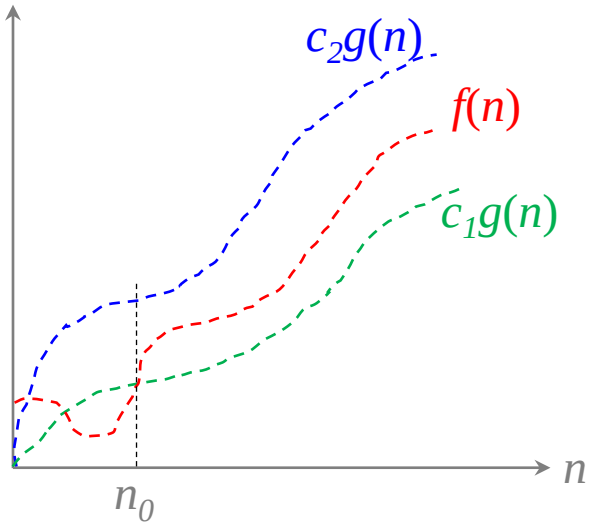
O(*n*)

A função *busca_chave* retorna o índice do nó inserido como argumento; se este não for encontrado, retorna nulo.
Para cada chamada à função, o algoritmo executa os testes: enquanto(*i* < *n*) e se ou senão(*L*[*i*] == *nó*).
A complexidade de pior caso para o algoritmo pode ser representada pela notação ***O*** (utilizada para limite assintótico superior), e para este algoritmo sua ordem seria → ***O***(*n*).

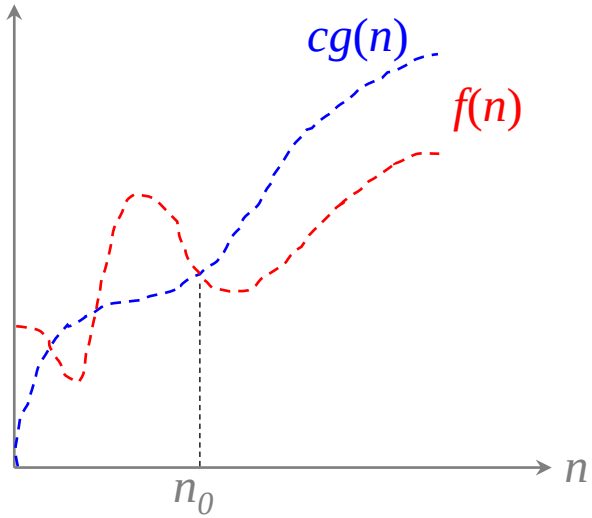
(Szwarcfiter & Markenzon, 2020)

Notação Θ e O

Uma $f(n) = \boldsymbol{\Theta}(g(n))$ implica $f(n) = \boldsymbol{O}(g(n)) \rightarrow$ em teoria de conjuntos escreve-se que $\boldsymbol{\Theta}(g(n)) \subseteq \boldsymbol{O}(g(n))$ (*téta* contido em *ó* grande)**.** Assim a prova de que qualquer função quadrática $\boldsymbol{a.n^2 + b.n + c}$, com $a > 0$, está em $\boldsymbol{\Theta}(\boldsymbol{n^2})$ assim como está em $O(\boldsymbol{n^2})$.



$f(n) = \boldsymbol{\Theta}(g(n))$



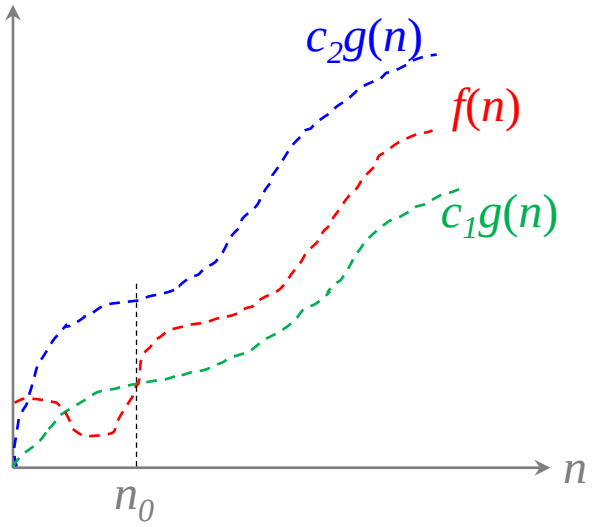
$f(n) = \boldsymbol{O}(g(n))$

(Szwarcfiter & Markenzon, 2020; Cormen et al., 2012)

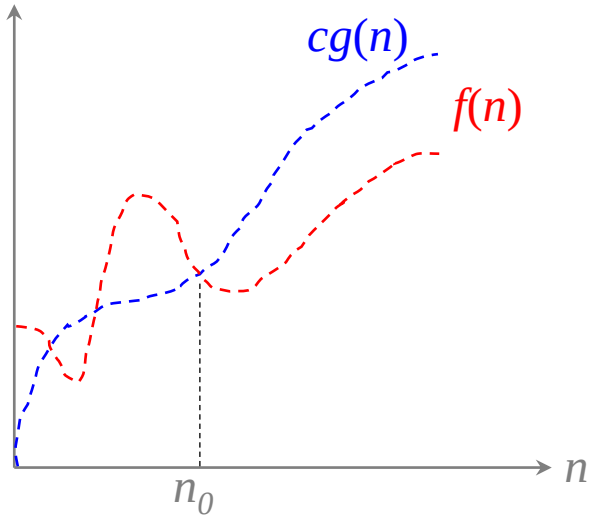
Notação Θ e O

Uma definição formal matemática pode justificar a intuição da simplificação citada anteriormente, tanto para a notação Θ como para a notação O, por exemplo:

$$a.n^2 + b.n + c = \Theta(n^2)$$
$$a.n^2 + b.n + c = O(n^2)$$



$f(n) = \boldsymbol{\Theta}(g(n))$



$f(n) = \boldsymbol{O}(g(n))$

(Szwarcfiter & Markenzon, 2020; Cormen et al., 2012)

Exercício:

- *Simular o algoritmo da função* “busca_linear” *da implementação a seguir, que faz a busca dos nós da lista;*
- *Implementar o código completo com a função* “busca_linear” *sendo executada uma vez;*
- *Implementar o código completo com a função* “busca_linear” *sendo executada mil vezes;*
- *Verificar a variação da dimensão de entrada e o impacto no tempo de execução do algoritmo.*

Lista_linear_sequencial_I.py

Lista linear em alocação sequencial → algoritmo de busca executado 1 vez

```python
"""
Created on Wed Jul 13 19:42:53 2022

@author: cmari
"""
import time

L = [1, "Goiânia", "Goiás", 2, "Bonito", "Mato Grosso", 3, "Carangola", "Minas Gerais", 4, "Peruíbe", "São Paulo", 20, "Serra", "Espírito Santo"]
n = len(L)
print ("dimensão da lista L = ", n)
print ("L[0]", L[0])
print ("L[3]", L[3])
print ("L[6]", L[6])
print ("L[9]", L[9])
print ("L[4]", L[4])
print ("L[4][2]", L[4][2])
print ("L[2][1]", L[2][1])
print("\n")
def busca_linear(x):
  for p in range(1):
       i = 0
       while i < n:
         if L[i] == x:
           return i
         else:
            i = i + 1
tempo_inicial = time.time()
print("índice da chave (5) = ", busca_linear(5))
print("índice da chave (1) = ", busca_linear(1))
print("índice da chave (4) = ", busca_linear(4))
print("índice da chave (3) = ", busca_linear(3))
print("índice da chave (2) = ", busca_linear(2))
print("índice da chave (0) = ", busca_linear(0))
print("índice da chave (20) = ", busca_linear(20))
tempo_final = time.time()
print("tempo de execução = ", tempo_final - tempo_inicial)
```

Console 1/A

```
In [23]: runfile('C:/Users/cmari/.spyder-py3/
Lista_linear_sequencial_I.py', wdir='C:/
dimensão da lista L =  15
L[0] 1
L[3] 2
L[6] 3
L[9] 4
L[4] Bonito
L[4][2] n
L[2][1] o


índice da chave (5) =  None
índice da chave (1) =  0
índice da chave (4) =  9
índice da chave (3) =  6
índice da chave (2) =  3
índice da chave (0) =  None
índice da chave (20) =  12
tempo de execução =  0.0
```

n = 15
x = 5
i = 0
enquanto (i < 15)
se L[i] = = x (L[0] = 1)
retorna i
senão
i = 1 + 1 .....
i = 5
L[5] = “Mato Grosso” ⇒ nó não achado

x = 1
i = 0
enquanto (i < 15)
se L[i] = = x (L[0] = 1)
retorna i = 0 ⇒ índice do nó = 1

Lista linear em alocação sequencial → algoritmo de busca executado n vezes

Lista_linear_sequencial_I.py

```
"""
Created on Wed Jul 13 19:42:53 2022

@author: cmari
"""
import time

L = [1, "Goiânia", "Goiás", 2, "Bonito", "Mato Grosso", 3, "Carangola", "Minas Gerais", 4, "Peruíbe", "São Paulo", 20, "Se
n = len(L)
print ("dimensão da lista L = ", n)
print ("L[0]", L[0])
print ("L[3]", L[3])
print ("L[6]", L[6])
print ("L[9]", L[9])
print ("L[4]", L[4])
print ("L[4][2]", L[4][2])
print ("L[2][1]", L[2][1])
print("\n")
def busca_linear(x):
  for p in range(1000):
        i = 0
        while i < n:
          if L[i] == x:
            return i
            #i = n + 1
          else:
             i = i + 1
tempo_inicial = time.time()
print("índice da chave (5) = ", busca_linear(5))
print("índice da chave (1) = ", busca_linear(1))
print("índice da chave (4) = ", busca_linear(4))
print("índice da chave (3) = ", busca_linear(3))
print("índice da chave (2) = ", busca_linear(2))
print("índice da chave (0) = ", busca_linear(0))
print("índice da chave (20) = ", busca_linear(20))
tempo_final = time.time()
print("tempo de execução = ", tempo_final - tempo_inicial)
```

Console 1/A

```
In [24]: runfile('C:/Users/cmari/.spyder-py3/
Lista_linear_sequencial_I.py', wdir='C:/Users/cmari/.spyder-py3')
dimensão da lista L =  15
L[0] 1
L[3] 2
L[6] 3
L[9] 4
L[4] Bonito
L[4][2] n
L[2][1] o


índice da chave (5) =  None
índice da chave (1) =  0
índice da chave (4) =  9
índice da chave (3) =  6
índice da chave (2) =  3
índice da chave (0) =  None
índice da chave (20) =  12
tempo de execução =  0.006198406219482422
```

Referências Bibliográficas

- Estruturas de Dados e Seus Algoritmos
  Jayme L. Szwarcfiter & Lilian Markenzon
  3ª edição – editora *gen* LTC – 2010 - 2020

- Matemática Avançada para Engenharia
  Dennis G. Zill & Michael R. Cullen
  Álgebra Linear e Cálculo Vetorial (2) – 3ª edição, editora Bookman – 2009

- Algoritmos – Teoria e Prática
  Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein.
  3ª edição – editora Elsevier - *gen* LTC – 2012