

Estruturas de dados

Noções de complexidade de algoritmos

Marco A L Barbosa

malbarbo.pro.br

Começando

- 1) Quais critérios podemos utilizar para escolher entre dois algoritmos diferentes que resolvem o mesmo problema?
- 2) O que é complexidade de algoritmo?
- 3) O que é análise de algoritmo?
- 4) Quais são as duas principais formas de análise de algoritmos?
- 5) Qual é o propósito da notação assintótica?
- 6) O que significa dizer que uma função $f(n)$ é $O(g(n))$?

Praticando

- 7) Considere o seguinte algoritmo que verifica que calcula a amplitude dos valores de uma lista não vazia

```
def amplitude(lst: list[int]) -> int
    ...

    Calcula a amplitude dos valores em *lst*, isto é, a diferença entre os
    valores máximo e mínimo.

    Exemplos
    >>> amplitude([3])
    0
    >>> amplitude([3, 1, 5, 2, 4])
    4
    ...

    assert len(lst) > 0
    min = lst[0]
    for i in range(1, len(lst)):
        if lst[i] < min:
            min = lst[i]
    max = lst[0]
    for i in range(1, len(lst)):
        if lst[i] > max:
            max = lst[i]
    return max - min
```

Para uma lista com n elementos, quantas vezes as operações $<$ e $>$ são executadas? Qual é a complexidade de tempo do algoritmo?

- 8) Considere o seguinte algoritmo que verifica se uma lista é palíndromo

```
def palindromo(lst: list[int]) -> bool:
    ...

    Produz True se *lst* é palíndromo, isto é, os elementos de *lst* são os
```

mesmo quando lidos da direita para a esquerda e da esquerda para a direita.

Exemplos

```
>>> palindromo([4, 1, 3, 3, 1, 4])
```

```
True
```

```
>>> palindromo([1, 2, 2, 2])
```

```
False
```

```
'''
```

```
eh_palin = True
```

```
i = 0
```

```
j = len(lst) - 1
```

```
while i < j and eh_palin:
```

```
    if lst[i] != lst[j]:
```

```
        eh_palin = False
```

```
    i = i + 1
```

```
    j = j - 1
```

```
return eh_palin
```

- Para uma lista de tamanho n , quantas vezes no mínimo a operação `!=` será executada? Como deve estar a entrada para que isso aconteça? Qual é a complexidade de tempo do algoritmo no melhor caso?
- Para uma lista de tamanho n , quantas vezes no máximo a operação `!=` será executada? Como deve estar a entrada para que isso aconteça? Qual é a complexidade de tempo do algoritmo no pior caso?

Avançando

- 9) O algoritmo de ordenação por inserção usa o método incremental para ordenar uma lista de valores. Quando o algoritmo está processando o elemento da posição i , a sub lista de 0 até $i-1$ já está ordenada, então o trabalho do algoritmo é mover o elemento da posição i de maneira a sub lista de 0 até i fique ordenada. A seguinte função implementa o algoritmo de ordenação por inserção.

```
def ordena_insercao(lst: list[int]):
```

```
    '''
```

```
    Ordena os elementos de *lst* em ordem não decrescente.
```

```
    Exemplos
```

```
>>> lst = [8, 1, 6, 3, 1]
```

```
>>> ordena_insercao(lst)
```

```
>>> lst
```

```
[1, 1, 3, 6, 8]
```

```
'''
```

```
for i in range(1, len(lst)):
```

```
    j = i
```

```
    while j > 0 and lst[j - 1] > lst[j]:
```

```
        t = lst[j - 1]
```

```
        lst[j - 1] = lst[j]
```

```
        lst[j] = t
```

```
        j = j - 1
```

- Para uma lista de tamanho n , quantas vezes no mínimo as comparações da linha do `while` serão executadas? Como deve estar a entrada para que isso aconteça? Qual é a complexidade de tempo do algoritmo no melhor caso?
- Para uma lista de tamanho n , quantas vezes no máximo as comparações da linha do `while` serão executadas? Como deve estar a entrada para que isso aconteça? Qual é a complexidade de tempo do algoritmo no pior caso?