Algoritmos e Estrutura de Dados

Aula 19 – Estrutura de Dados: Quick Sort

Prof. Tiago A. E. Ferreira

Quick Sort

- Criado em 1960 por Hoare.
- Segue a mesma abordagem do algoritmo Mergesort, ou seja, utiliza a técnica dividir e conquistar.
- Basicamente é o mesmo algoritmo, mas em vez de separar no meio a lista a ser ordenada, o algoritmo usa outro método se separação.

Quick Sort

- A divisão do Quicksort é feita em elementos maiores e menores de um elemento especial chamado **pivot.**
- Este pivot é o elemento chave da divisão do quicksort.
- Quanto mais eficiente é a escolha do pivot, menor será o custo de ordenação.
- Um dos problemas iniciais é, então, encontrar o pivot.

Quick Sort

- Algumas sugestões de pivot:
  - Primeiro elemento
  - Último elemento
  - Elemento central (Mergesort)
  - Aleatório
  - Usando critérios específicos

O Algoritmo

- Escolha um pivot;
- Reordene a lista de forma que são menores que o pivot fiquem antes; e
- Os elementos maiores fiquem depois
- Os iguais independem de lado
- Coloque o pivot na posição final
- Recursivamente ordene a sublista dos menores elementos e a dos maiores.

Algoritmo 1:

```
def qsort1(list):
    """ Quicksort using list comprehensions >>>
    qsort1<<docstring test numeric input>> <<docstring test
    numeric output>> >>> qsort1<<docstring test string input>>
    <<docstring test string output>> """
    if list == []:
        return []
    else:
        pivot = list[0]
        lesser = qsort1([x for x in list[1:] if x < pivot])
        greater = qsort1([x for x in list[1:] if x >= pivot])
    return lesser + [pivot] + greater
```

Algoritmo 2:

```
def qsort2(list):
    """ Quicksort using a partitioning function >>>
    qsort2<<docstring test numeric input>> <<docstring test
    numeric output>> >>> qsort2<<docstring test string
    input>> <<docstring test string output>> """
    if list == []:
        return []
    else:
        pivot = list[0]
        lesser, equal, greater = partition(list[1:], [], [pivot], [])
    return qsort2(lesser) + equal + qsort2(greater)
```

Algoritmo 3

```
def partition(list, l, e, g):
    if list == []:
        return (l, e, g)
    else:
        head = list[0]
        if head < e[0]:
            return partition(list[1:], l + [head], e, g)
        elif head > e[0]:
            return partition(list[1:], l, e, g + [head])
        else:
            return partition(list[1:], l, e + [head], g)
```

Função Partição

```
def partition(list, l, e, g):
    while list != []:
        head = list.pop(0)
        if head < e[0]:
            l = [head] + l
        elif head > e[0]:
            g = [head] + g
        else:
            e = [head] + e
    return (l, e, g)
```

Escolha Aleatória do Pivot

```
from random import randrange
def qsort1a(list):
    """ Quicksort using list comprehensions and randomized
    pivot >>> qsort1a<<docstring test numeric input>>
    <<docstring test numeric output>> >>>
    qsort1a<<docstring test string input>> <<docstring test
    string output>> """
    def qsort(list):
        if list == []:
            return []
        else:
            pivot = list.pop(randrange(len(list)))
            lesser = qsort([l for l in list if l < pivot])
            greater = qsort([l for l in list if l >= pivot])
    return lesser + [pivot] + greater return qsort(list[:])
```

Complexidade

- No pior casso:
  - $O(n^2)$
- No caso médio:
  - O(nlogn)

Exercício:

- Implemente o QuickSort nas suas diversas versões e teste-o com uma seqüência de números inteiros.