Busca sequencial e binária

Objetivo

Ao final dessa aula você vai saber escrever algoritmos eficientes para realizar buscas em listas.

Tópicos

- Busca sequencial;
- Busca binária ou Busca em vetor ordenado.

Algoritmos de Busca:

Algoritmos de busca verificam se uma dada informação ocorre em uma sequência ou não. Por exemplo, dada uma sequência de números guardados em uma lista *seq* e um número *x*, escreva uma função que responda à pergunta: *x ocorre na sequência?*

Uma possível solução é percorrer a lista toda variando o índice *i* de 0 a *len(seq)-1* e comparando cada elemento *seq[i]* com *x*. Caso o valor seja encontrado a função retorna *True* e, caso contrário, retorna *False*. Essa solução é conhecida como **Busca Sequencial.**

```
1 def busca_sequencial( seq, x):
2     '''(list, float) -> bool'''
3     for i in range(len(seq)):
4         if seq[i] == x:
5             return True
6     return False
7
8
```

ActiveCode: 1 (algoritmo_de_busca_sequencial)

Run Save Load

Exercício 23.1

Escreva um programa que leia uma sequência com *N* números reais e imprime a sequência eliminando os elementos repetidos. Esse exercício pode ser dividido em 2 partes:

Parte A

Escreva a função:

```
1 def acha(seq, x):
2     ''' (list, float) -> int
3         retorna a posicao em que x ocorre na list
4         '''
5     # escreva a funcao
6
7
```

ActiveCode: 2 (ex23_1_parte_A)

Run Save Load

** Parte B**

```
def main():
    ''' programa que le uma sequencia com N eleme
        sem repeticoes.
        '''

    # escreva o programa

main()
```

ActiveCode: 3 (ex23_1_parte_B)

Run Save Load

Exercício 23.2

Quando utilizamos o algoritmo de busca sequencial para procurar um elemento de valor *x* em uma sequência *seq*, toda a sequência precisa ser varrida quando *x* não está presente em *seq*.

Para criarmos um algoritmo mais eficiente, vamos assumir que a sequência esteja em ordem alfabética, como em um dicionário. Nesse caso, ao invés de testar um elemento de cada vez sequencialmente, podemos aplicar o seguinte algoritmo:

- considere o elemento *M*, no meio da lista.
- caso *x* for igual a *M*, então a busca termina pois encontramos o valor procurado.
- caso *M* for maior que *x*, então *x* deve estar na primeira metade da sequência. A busca deve continuar **apenas** nessa metade. Mas se o comprimento dessa metade for nulo, a busca deve termina e o valor não foi encontrado.
- caso *M* for menor que *x*, então *x* deve estar na segunda metade da sequência. A busca deve continuar **apenas** nessa metade. mas se o comprimento dessa metade for nulo, então a busca termina e o valor não foi encontrado.

Esse algoritmo é conhecido como **Busca Binária** pois a cada iteração metade da sequência é eliminada da busca. Dessa forma, usando o algoritmo de busca sequencial em uma sequência com 1024 elementos, todos os 1024 elementos devem ser testados antes do algoritmo indicar que o elemento não está na lista. No caso da busca binária, o primeiro teste elimina 512 elementos, o segundo 256, o terceiro 128, e depois 64, 32, 16, 8, 4, 2, até que a lista contenha apenas 1 elemento. Dessa forma, ao invés de 1024, apenas 10 elementos (ou *log(len(seq))*) precisam ser testados.

```
def busca_binaria(seq, x):
    ''' (list, float) -> bool
        retorna a posicao em que x ocorre na li
        ou None caso contrario, usando o algori
        '''
    # escreva a sua funcao
    return None


# escreva alguns testes da funcao busca_binaria
seq = [4, 10, 80, 90, 91, 99, 100, 101]
testes = [80, 50]
```

```
13
14 for t in testes:
15     pos = busca_binaria(seq, t)
16     if pos is None:
17         print("Nao achei ", t)
18     else:
19         print("Achei ", t)
20
```

ActiveCode: 4 (ex23_2_busca_binaria)

Run Save Load

Aulas de Introdução à Computação com Python »