Rafael Vieira Coelho

rafaelvc2@gmail.com

12 - Algoritmos de Busca

PARTE 1

Busca Sequencial e Binária

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

Tópicos:

- Busca sequencial;
- Busca binária (ordenado).

python™

Algoritmos de Busca: Busca Sequencial

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 vet e um número num, escreva uma função que responda à pergunta: num ocorre na sequência?

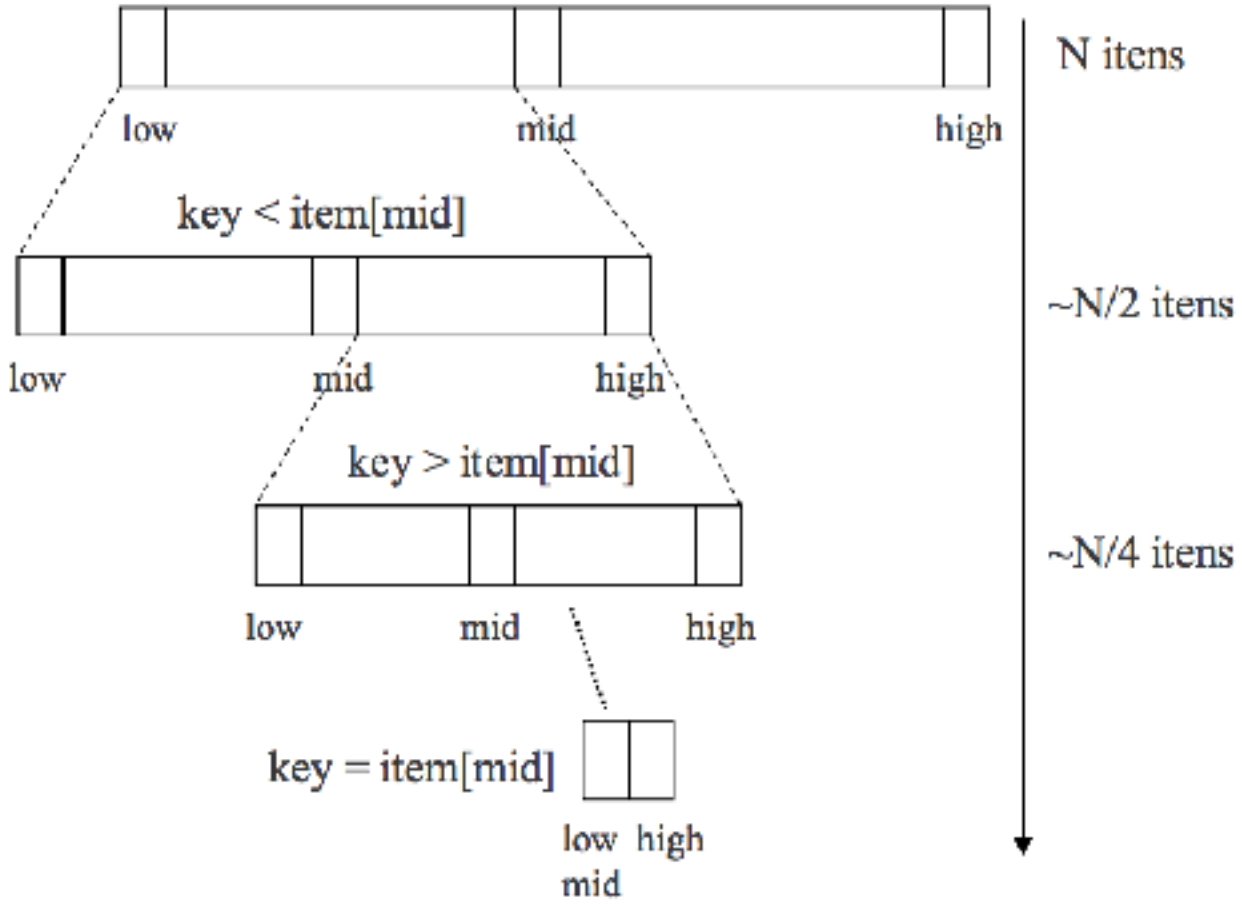
Uma possível solução é percorrer a lista toda variando o índice i de 0 a len(vet) e comparando cada elemento vet[i] com num. Caso o valor seja encontrado a função retorna True e, caso contrário, retorna False. Essa solução é conhecida como Busca Sequencial.

```
#busca sequencial
def sequential_search(vet, num):
    for i in range(len(vet)):
        if (vet[i] == num):
            return True
    return False
```

python™

Busca Binária

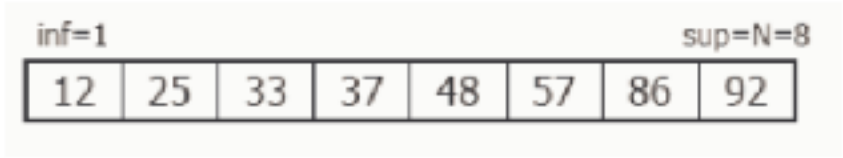
A ideia é dividir a lista em pedaços menores (sempre ao meio) até achar o valor procurado.



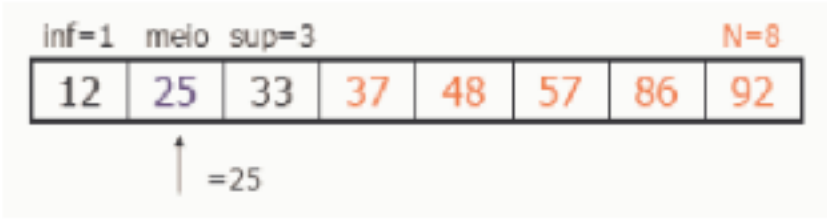
Busca Binária

Para pesquisar o número 25 na lista ao lado:

Busca-se por 25



Busca-se por 25



Em cada passo, o tamanho do arranjo em que se busca é dividido por 2

Busca Binária

```python
# busca binária iterativa
def binary_search_ite(vet, num):
    esquerda = 0
    direita = len(vet)
    tentativa = 1
    while 1:
        meio = (esquerda + direita) // 2
        aux_num = vet[meio]
        if num == aux_num:
            return tentativa
        elif num > aux_num:
            esquerda = meio
        else:
            direita = meio
        tentativa += 1
```

python™

Busca Binária Recursiva

```
def binary_search_rec(vet, num, esq, dir, tentativa):
        meio = (esq + dir) // 2
        aux_num = vet[meio]
        if num == aux_num:
                return tentativa
        elif num > aux_num:
                return binary_search_rec(vet, num, meio, dir, tentativa + 1)
        return binary_search_rec(vet, num, esq, meio, tentativa + 1)
```

Teste

```
def teste():
        vet = [i for i in range(1, 1000001)]
        num = random.choice(vet)
        print('Numero escolhido: %d' % num)
        print('Tentativa (iterativo): %d' % binary_search_ite(vet, num))
        print('Tentativa (recursivo): %d' % binary_search_rec(vet, num, 0, len(vet), 1))

teste()
```

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:

```
def acha(seq, x):
    ''' (list, float) -> int
        retorna a posicao em que x ocorre na lista, ou None caso contrario
        '''
    # escreva a funcao


```

ActiveCode: 2 (ex23_1_parte_A)

Run Save Load

** Parte B**

```
def main():
    ''' programa que le uma sequencia com N elementos e a imprime
        sem repeticoes.
        '''

    # escreva o programa

main()


```

ActiveCode: 3 (ex23_1_parte_B)

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.

```
1  def busca_binaria(seq, x):
2      ''' (list, float) -> bool
3          retorna a posicao em que x ocorre na lista ordenada,
4          ou None caso contrario, usando o algoritmo de busca binaria.
5          '''
6      # escreva a sua funcao
7      return None
8
9
10 # escreva alguns testes da funcao busca_binaria
11 seq = [4, 10, 80, 90, 91, 99, 100, 101]
12 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
21
```