gianfrancojoel_condoriluna_aula2Pratico

August 20, 2024

1 Aula 02 - Exercício prático análise de algoritmos recursivos

1.1 1) (1,0) Proponha um algoritmo para busca sequencial e outro para busca binária.

1.1.1 a) Execute ambas as funções para encontrar o número 9 e 99.999 num vetor aleatório de 100.000 elementos (use o mesmo vetor em ambos os códigos).

1.1.2 b) Verifique qual dos dois executou mais rápido (use uma time() function).

1.1.3 c) Faça a análise de complexidade de tempo de ambos os algoritmos.

1.1.4 SOLUÇÃO:

Entendo que nesse problema o importante é o tempo que leva para uma busca sequencial e uma busca binária, então baseado nessa hipótese acho que deveria criar um vetor ordenado de 1 a 100.000 pois não faria sentido criar um vetor aleatório e depois retornar para ordenar esse vetor para que depois simplesmente se pase para os algoritmos de busca.

O tempo que leve para ordenar seria o mesmo para ambos algoritmos, portanto, poderia fornecer o vector ordenado para ambos algoritmos e comparar seu tempo.

Também por ter o vetor ordenado eu teria certeza que o número que está na posição 9 e 99999 são iguais para ambos os algoritmos.

Etapas de resolução do problema:

- Crie um vetor ordenado de 1 a 100.000.
- Crie um algoritmo de busca sequencial.
- Crie um algoritmo de busca binária.

[18]:
```python
#Python version 3.10.12

import time

# Gerando um vetor aleatório de 100.000 elementos
tamanho_vetor = 100000
vetor = list(range(1, tamanho_vetor+1))

# Algoritmo de busca sequencial
def busca_sequencial(vetor, alvo):
    for i in range(len(vetor)):
```

```
        if vetor[i] == alvo:
            return i
    return -1

# Algoritmo de busca binária
def busca_binaria(vetor, alvo):
    p_esquerdo, p_direito = 0, len(vetor) - 1
    while p_esquerdo <= p_direito:
        p_meio = (p_esquerdo + p_direito) // 2
        if vetor[p_meio] == alvo:
            return p_meio
        elif vetor[p_meio] < alvo:
            p_esquerdo = p_meio + 1
        else:
            p_direito = p_meio - 1
    return -1
```

[19]:
```
# Números a serem buscados
numeros_para_buscar = [9, 99999]

# Executando e medindo o tempo para busca sequencial
tempos_sequencial = {}
for numero in numeros_para_buscar:
    t_inicio = time.time()
    busca_sequencial(vetor, numero)
    tempos_sequencial[numero] = time.time() - t_inicio

# Executando e medindo o tempo para busca binária
tempos_binaria = {}
for numero in numeros_para_buscar:
    t_inicio = time.time()
    busca_binaria(vetor, numero)
    tempos_binaria[numero] = time.time() - t_inicio
```

[20]:
```
print("Algoritmo de busca sequencial")
print("* O tempo que levou para encontrar o número 9 foi     :␣
 ↪",tempos_sequencial[9])
print("* O tempo que levou para encontrar o número 99.999 foi:␣
 ↪",tempos_sequencial[99999])
print("Algoritmo de busca binária")
print("* O tempo que levou para encontrar o número 9 foi     :␣
 ↪",tempos_binaria[9])
print("* O tempo que levou para encontrar o número 99.999 foi:␣
 ↪",tempos_binaria[99999])
```

```
Algoritmo de busca sequencial
* O tempo que levou para encontrar o número 9 foi     :  3.814697265625e-06
```

```
* O tempo que levou para encontrar o número 99.999 foi:  0.008069276809692383
Algoritmo de busca binária
* O tempo que levou para encontrar o número 9 foi     :  9.5367431640625e-06
* O tempo que levou para encontrar o número 99.999 foi:  4.291534423828125e-06
```

Complexidade do algoritmo de busca sequencial `for i in range(len(vetor)):` -> O(n)

`if vetor[i] == alvo:` -> O(1)

`return i` -> O(1)

`return -1` -> O(1)

$ $\boxed{Resposta: \quad O(n)}$ $

Complexidade do algoritmo de busca binária

- Revendo o código podemos ver que em cada iteração ele será convertido em $\frac{n}{2}$, $\frac{n}{4}$, $\frac{n}{8}$ e assim por diante até chegando a 1.
- Então o número de iterações necessárias para reduzir o tamanho do vetor para 1 é $\log_2(n)$

$ $\boxed{Resposta: \quad O(\log(n))}$ $

1.1.5 Conclusões:

- O algoritmo de busca sequencial é mais rápido para buscar um número que está em uma posição próxima ao início mas é muito caro buscar um número que está quase no final do vetor, mais longe mais tempo.
- O algoritmo de busca binária é mais lento que o algoritmo de busca sequencial para encontrar um elemento que esteja próximo do início do vetor, mas é muito mais rápido para encontrar um elemento que esteja próximo do final do vetor.
- Com isso podemos concluir que o custo de tempo do algoritmo de busca binária depende logaritmicamente da quantidade de elementos que o vetor possui, pois não há muita diferença entre buscar um elemento independente da posição em que ele se encontra, algo que é não acontece com o algoritmo sequencial que depende muito da quantidade de elementos do vetor e da posição em que se encontra o elemento a ser pesquisado.