
Detonando no Code Challenge
Com Python e Java
Cleuton Sampaio
DESENVOLVEDOR
GURU

Complexidade de Espaço

Faça este desafio. Leia a descrição, pause o vídeo e depois retorne para conferir.
Um método recebe um vetor ordenado de inteiros e um número a encontrar. Deve retornar a posição (começando em zero) onde o número se encontra no vetor.
Exemplo:

- [1,3,6,10,12,25] número a encontrar: 12 → Resultado: 4

Restrições: O programa deve executar no menor tempo e espaço possível.

Pause o vídeo agora!

Solução simples

```python
class Linear:
    def busca (self, nums, target):
        for i in range(len(nums)):
            if target == nums[i]:
                return i
        return None

if __name__=="__main__":
    l = Linear()
    print(l.busca([1,3,6,10,12,25],12))
```

```java
public class Busca_simples {
        public int busca (int [] nums, int target) {
                for (int i=0; i<nums.length; i++) {
                        if (nums[i] == target) {
                                return i;
                        }
                }
                return -1;
        }
        public static void main(String[] args) {
                System.out.println((new Busca_simples())
                                .busca(new int [] {1,3,6,10,12,25},
12));
        }
}
```

Tempo: **O(n)**
Espaço: **O(n)**

Resolvem o problema, mas não são as implementações de menor tempo possível. Usar um hashmap ou Dictionary, como fizemos, **violaria a restrição de espaço**.

Busca binária

Um algoritmo que usa a estratégia “Divide and conquer” que divide o vetor em metades:

Há implementações iterativas ou recursivas.

No pior caso a complexidade de tempo é O(log n) que é menor que O(n).

A complexidade de espaço é a mesma.

```
BUSCA-BINÁRIA(V[], início, fim, e)
    i recebe o índice do meio entre início e fim
    se (v[i] = e) entao
        devolva o índice i   # elemento e
encontrado
    fimse
    se (inicio = fim) entao
        não encontrou o elemento procurado
    senão
      se (V[i] vem antes de e) então
        faça a BUSCA-BINÁRIA(V, i+1, fim, e)
      senão
        faça a BUSCA-BINÁRIA(V, inicio, i-1, e)
      fimse
    fimse
```

Busca binária

```
from bisect import bisect_left
class Binary:
    def busca (self, nums, target):
        i = bisect_left(nums, target)
        if i != len(nums) and nums[i] == target:
            return i
        return None
if __name__=="__main__":
    l = Binary()
    print(l.busca([1,3,6,10,12,25],12))
```

Assumindo que os vetores estão em ordem ascendente, tanto python como java possuem bibliotecas para busca binária:
- Python: bisect.bisect_left();
- Java: java.util.Arrays.binarySearch();

Complexidade de tempo no pior caso é O(log n);
Complexidade de espaço depende:
- Recursiva: O(log n);
- Iterativa: O(1);

```
import java.util.Arrays;
public class Busca_binaria {
	public int busca (int [] nums, int target) {
		return Arrays.binarySearch(nums,target);
	}
	public static void main(String[] args) {
		System.out.println((new Busca_binaria())
				.busca(new int [] {1,3,6,10,12,25}, 12));
	}
}
```

Medindo a complexidade de espaço

Complexidade de espaço: Espaço auxiliar + espaço para variáveis de entrada

O(1): O espaço independe da entrada. Não há loops nem recursões;
O(n): O espaço depende da entrada. Pode haver loops;

Se você está passando um vetor como um parâmetro, pode haver complexidade de espaço maior, dependendo da forma da passagem:

- Por referência: O(1) – pode ser desprezado;
- Por valor: O(n) – É feita uma cópia do vetor;

Em python os objetos são passados por referência e em Java o valor passado é uma referência ao objeto real, então consideramos a passagem do vetor como O(1).

Se tivéssemos criado um hash (dictionary ou hashmap) em vez de usar busca binária, a complexidade seria O(2n), que seria O(n). Mas estaríamos utilizando espaço de memoria maior ao criar um novo vetor.