

Ternary Search

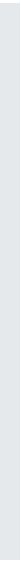
Algoritmo de Busca

Arthur Moro

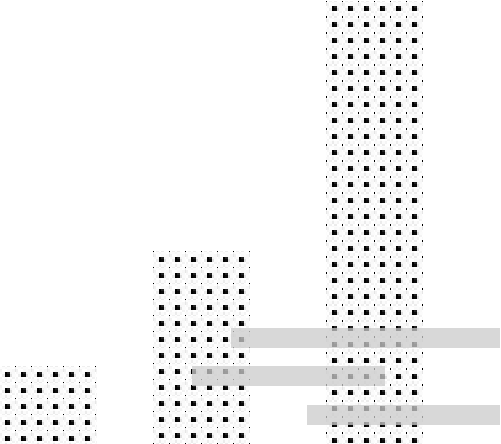


Algoritmos de Busca

O que são:



Algoritmos de busca são desenvolvidos para checar ou recuperar um elemento de uma base de dados. São normalmente categorizados em:

- **Busca Sequencial:** a lista é atravessada sequencialmente e todos os elementos são checados.
 - **Busca em Intervalos:** funciona apenas em listas já organizadas, dividindo o espaço de busca em pedaços.
- 

Positivos e Negativos



Eficiência

Ternary search reduz o espaço de busca em um terço a cada iteração, o fazendo mais rápido do que Linear search, e mais rápido do que Binary search em certos casos.

Busca Rápida

Ternary search possui um tempo de busca de $O(\log N)$, funcionando melhor em busca de strings.

Uso de Memória Eficiente

Ternary search armazena apenas os caracteres das keys pertencentes aos nodos.

Precisa de Dados Organizados

Ternary search só é eficiente em dados organizados, sendo mandatório que o array seja organizado antes do começo da busca.

Mais Comparações

Ternary search pode, no pior dos casos, fazer 1.5 vezes mais comparações de keys do que Binary search.

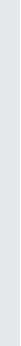
Implementação Complexa

Ternary search é mais complicado de se implementar comparado com Binary search, por conta dos cálculos e comparações adicionais necessárias

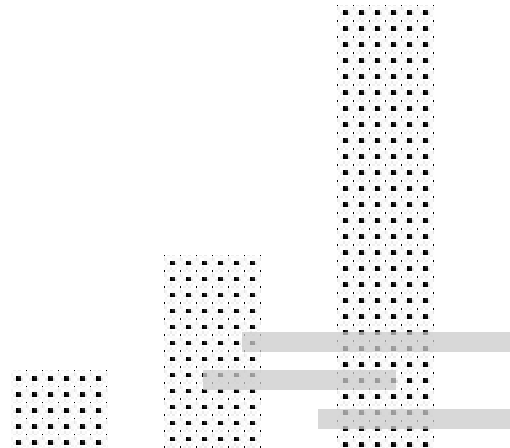
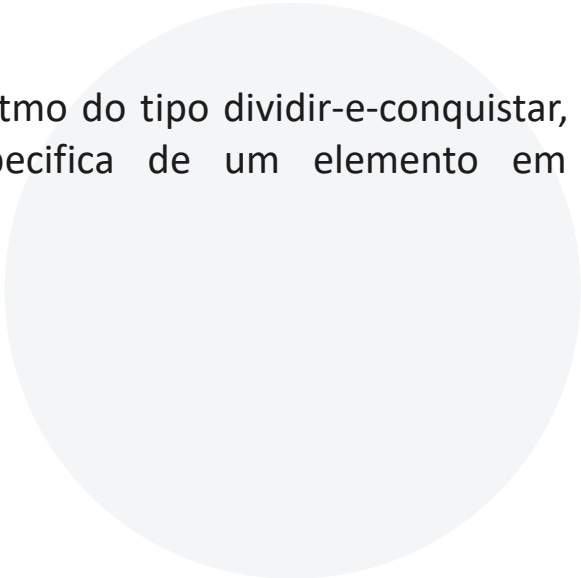


Ternary Search

Como funciona:



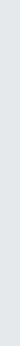
Ternary search é um algoritmo do tipo dividir-e-conquistar, usado para encontrar a posição específica de um elemento em um array organizado.



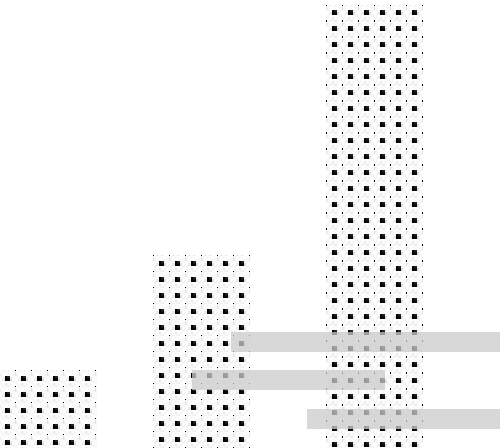
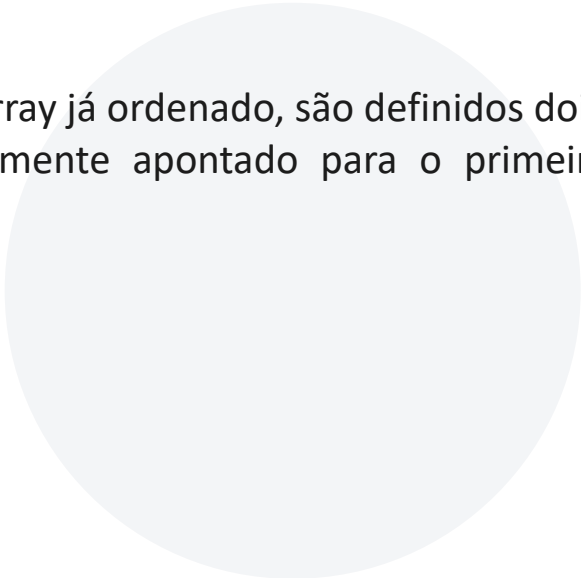


Ternary Search

Como funciona:



1 - **Inicialização:** Com um array já ordenado, são definidos dois ponteiros, esquerdo e direito, inicialmente apontado para o primeiro e ultimo elementos do array.



Ternary Search

Como funciona:

2 - **Divisão do Array:** Calcule dois pontos médios, `meio1` e `meio2`, dividindo o espaço de busca em 3 partes com numero de elementos similar.

$$\text{meio1} = \text{esquerda} + (\text{direita} - \text{esquerda}) / 3$$
$$\text{meio2} = \text{direita} - (\text{direita} - \text{esquerda}) / 3$$

O array agora esta dividido em `[esquerda, meio1]`, `(meio1, meio2)`, e `[meio2, direita]`.

Ternary Search

Como funciona:

3 - Comparação com o alvo:

- Se o alvo é igual ao elemento em meio1 ou meio2, a busca foi concluída e o index é retornado.
- Se o alvo é menor que o elemento em meio1, atualizar o ponteiro direito para meio1 - 1.
- Se o alvo é maior que o elemento em meio2, atualizar o ponteiro esquerdo para meio2 + 1.
- Se o alvo esta entre os elementos em meio1 e meio2, atualizar o ponteiro esquerdo para meio1 + 1 e o ponteiro direito para meio2 - 1.



Ternary Search

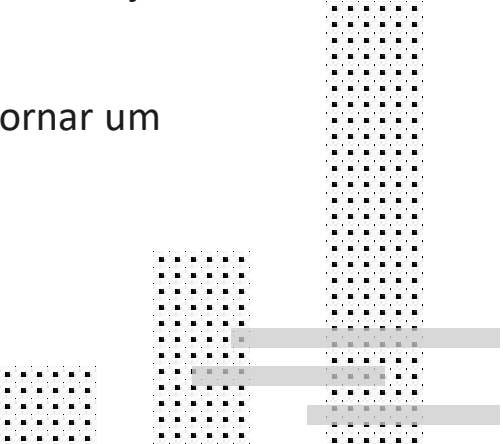
Como funciona:



4 - Repetir ou Concluir a busca:

Repetir o processo com o espaço de busca reduzido até que o alvo seja encontrado ou o espaço de pesquisa se torne vazio.

Se o espaço de busca esta vazio e o alvo não foi encontrado, retornar um valor indicando que o alvo não esta presente no array.



Ternary Search

Implementação
em C#:

```
0 references
5 public class Program
6 {
7     //Função para realizar a busca
8     5 references
9     public static int TernarySearch(int esquerda, int direita, int chave, int[] lista)
10    {
11        if (direita >= esquerda)
12        {
13            //Para encontrar meio1 e meio2
14            int meio1 = esquerda + (direita - esquerda) / 3;
15            int meio2 = direita - (direita - esquerda) / 3;
16
17            //Checar se a chave esta em algum dos meios
18            if (lista[meio1] == chave)
19            {
20                return meio1;
21            }
22
23            if (lista[meio2] == chave)
24            {
25                return meio2;
26            }
27        }
28    }
29 }
```

Ternary Search

Implementação
em C#:

```
26
27     //Checar em qual região a chave está, e então repetir a busca.
28     if (chave < lista[meio1])
29     {
30         return TernarySearch(esquerda, meio1 - 1, chave, lista);
31     }
32     else if (chave > lista[meio2])
33     {
34         return TernarySearch(meio2 + 1, direita, chave, lista);
35     }
36     else
37     {
38         return TernarySearch(meio1 + 1, meio2 - 1, chave, lista);
39     }
40 }
41 //Retorna -1 caso a chave não seja encontrada
42 return -1;
43 }
```