Algoritmos de Busca

Algoritmos de busca são amplamente utilizados na programação para encontrar elementos em uma estrutura de dados. Duas estruturas de dados comumente usadas para implementar algoritmos de busca são a lista e a árvore binária. Vamos descrever a utilização, vantagens e desvantagens de cada uma delas.

Lista

Uma lista é uma *estrutura de dados* linear que armazena elementos sequencialmente. Cada elemento da lista possui uma referência para o próximo elemento (em uma lista encadeada simples) ou para o elemento anterior e próximo (em uma lista duplamente encadeada). As listas permitem a inserção, remoção e acesso aos elementos de forma eficiente. Os elementos não precisam estar ordenados, e o acesso aos elementos pode ser feito por índice (posição) ou percorrendo a lista sequencialmente. É possível ter elementos repetidos em uma lista.

Busca em uma lista:

Utilização: A busca em uma lista é feita percorrendo cada elemento da lista, comparando-o com o valor procurado até encontrar uma correspondência ou percorrer todos os elementos da lista.

Vantagens:

- Simplicidade: A implementação da busca em uma lista é relativamente simples.
- Baixo custo de inserção: A inserção de elementos em uma lista é geralmente eficiente.

Desvantagens:

- Baixa eficiência em grandes conjuntos de dados: A busca em uma lista requer percorrer todos os elementos, o que pode ser ineficiente para grandes conjuntos de dados. Sua complexidade de tempo é dada por O(n), onde n é o número de elementos na lista.

Árvore Binária

Uma árvore binária é uma *estrutura de dados* hierárquica composta por nós. Cada nó possui uma referência para um nó pai (exceto o nó raiz) e, no máximo, duas referências para os nós filhos esquerdo e direito. A árvore é dividida em subárvores, onde cada subárvore também é uma árvore binária. Diferentemente das listas, as árvores binárias possuem uma organização específica dos nós. Por exemplo, em uma árvore binária de busca, os nós são organizados de forma que os valores menores estão no lado esquerdo e os valores maiores estão no lado direito. As árvores binárias permitem busca, inserção e remoção eficientes de elementos, especialmente quando a árvore está balanceada. Elas são frequentemente utilizadas para representar hierarquias, como a estrutura de diretórios de um sistema operacional. Existem diferentes tipos de árvores binárias, como árvores binárias de busca, árvores AVL, árvores rubro-negras, entre outras, cada uma com características específicas de balanceamento e eficiência.

Busca em uma árvore binária:

Utilização: A busca em uma árvore binária é feita comparando o valor procurado com o valor do nó atual e percorrendo a árvore de acordo com o resultado da comparação, até encontrar o valor desejado ou chegar a um nó vazio.

Vantagens:

- Eficiência em grandes conjuntos de dados ordenados: A busca em uma árvore binária tem uma complexidade de tempo de O(log n) no caso médio, o que a torna eficiente para grandes conjuntos de dados ordenados.
- Facilidade de inserção e remoção: A estrutura da árvore binária permite uma fácil inserção e remoção de elementos.

Desvantagens:

- Complexidade de construção: A construção de uma árvore binária balanceada pode exigir tempo e recursos consideráveis.
- Árvore desbalanceada: em algumas situações, podemos ter o pior dos casos, quando a árvore não está balanceada e a complexidade de tempo passa a ser O(n). Isso pode acontecer quando você segue adicionando nós que são sempre maiores do que o nó anterior (seu pai). A mesma situação pode acontecer quando você sempre adiciona nós com valores inferiores aos de seus nós pai.

Resultados e Discussão

Para a verificação prática do desempenho de cada um dos métodos de busca apresentados, foram realizados testes de execução medindo a velocidade de processamento na busca de um elemento em um arranjo com mil elementos. Os resultados obtidos podem ser verificados nas Tabelas 1 e 2.

Tabela 1: Tempo de execução durante a busca do menor elemento do arquivo csv original

Simulação	Métodos de Busca	
	Lista	Árvore Binária
1	0,000074	0,000014
2	0,000068	0,000019
3	0,000067	0,000015
4	0,000074	0,000016
5	0,000068	0,000015
Média (s)	**0,000070**	**0,000016**

Para encontrar o menor elemento da lista, utilizamos uma variável auxiliar '*menor'* que recebe, a priori, o elemento da posição 0. Ao percorrer a lista é realizada uma verificação, comparando o elemento da variável auxiliar com o elemento da lista, atualizando a variável auxiliar sempre que um valor menor é encontrado. Desse modo, ao fim da lista o menor valor terá sido encontrado.

Já no caso da Árvore Binária, como o processo de ordenação é realizado ao inserir os elementos na árvore, a busca torna-se mais eficiente pois já sabemos a posição do menor elemento. Para encontrar o menor elemento da árvore binária, basta acessar o elemento mais à esquerda da árvore. Isso faz com que o algoritmo de busca não precise percorrer todos os elementos do arranjo, tornando a árvore mais eficiente que a lista.

Tabela 2: Tempo de execução durante a busca do maior elemento do arquivo csv original

Simulação	Métodos de Busca	
	Lista	Árvore Binária
1	0,000070	0,000016
2	0,000069	0,000018
3	0,000070	0,000016
4	0,000066	0,000014
5	0,000069	0,000015
Média (s)	**0,000069**	**0,000016**

A busca pelo maior elemento ocorre de maneira análoga ao caso anterior, a lista é percorrida por inteiro, onde a variável auxiliar *'maior'* é comparada com todos os elementos. Já na árvore binária, basta acessar o elemento mais à direita da árvore, de modo que o algoritmo de busca não precise percorrer todos os elementos do arranjo.

Em um segundo momento, foram realizados os testes com os elementos ordenados em ordem crescente, os resultados do desempenho de cada um dos métodos de busca podem ser observados nas Tabelas 3 e 4.

Tabela 3: Tempo de execução durante a busca do menor elemento do arquivo csv ordenado

Simulação	Métodos de Busca	
	Lista	Árvore Binária
1	0,000063	0,000008
2	0,000068	0,000009
3	0,000063	0,000008
4	0,000068	0,000008
5	0,000067	0,000008
Média (s)	**0,000066**	**0,000008**

Considerando que os elementos da lista estão ordenados de modo crescente, o primeiro elemento já é o menor valor. É possível notar uma pequena variação no tempo de execução do algoritmo. Esta variação pode ser ocasionada tendo em vista que ele não precisará atualizar o valor da variável

auxiliar em nenhum momento. Na árvore binária, o elemento mais à esquerda da estrutura é o próprio nó raiz, o que resulta no baixo tempo de execução do algoritmo.

Tabela 4: Tempo de execução durante a busca do maior elemento do arquivo csv ordenado

Simulação	Métodos de Busca	
	Lista	Árvore Binária
1	0,000098	0,000025
2	0,000095	0,000026
3	0,000093	0,000027
4	0,000102	0,000023
5	0,000095	0,000022
Média (s)	**0,000097**	**0,000025**

Já na busca pelo maior elemento com os dados ordenados de modo crescente, como observado na Tabela 4, temos o pior caso para ambos os métodos. A lista deverá percorrer todos os elementos, sempre comparando e atualizando o valor da variável auxiliar, o que justifica o aumento no tempo de execução. Já na árvore binária, todos os elementos estão posicionados à direita do elemento pai. Desse modo, para chegar ao maior elemento do conjunto de dados é preciso percorrer todos os elementos, influenciando no aumento do seu tempo de execução.