Google olabi

<ESTRUTURASDEDADOS COMPLEXAS>

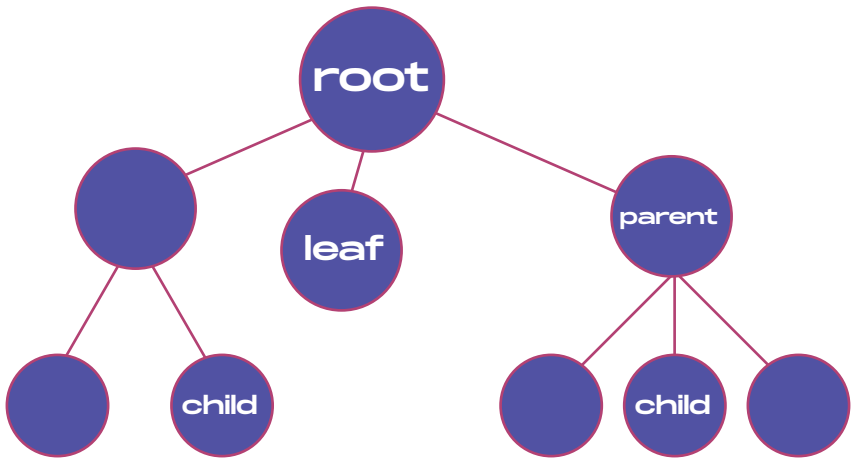
{ Estrutura de dados complexas

Nas aulas anteriores, vimos que as estruturas de dados são usadas para armazenar e organizar os dados. Além disso, através de algoritmos podemos manipular e usar essas estruturas de dados. Os tipos de estruturas já abordados foram Listas, Pilhas e Filas, que são estruturas de dados lineares. Nesta aula, iniciaremos o estudo das estruturas de dados não lineares.

{ Árvores

Árvores, ou Trees, são estruturas de dados não lineares, frequentemente usadas para representar dados hierárquicos. Estas estruturas de dados são baseadas em listas encadeadas que possuem um nó superior também chamado de raiz que aponta para outros nós, chamados de nós filhos, que podem ser pais de outros nós.

O nó mais alto é chamado raiz da árvore (root). Os elementos que estão diretamente sob um elemento são chamados de seus filhos (child). O elemento intermediário é chamado de seu pai (parent). Finalmente, o elemento sem filhos é chamado de folha (leaf).



Outros componentes destas estruturas de dados estão listados a seguir:

Subárvore: uma subárvore é uma árvore menor, mantida dentro de uma árvore maior. A raiz dessa árvore pode ser qualquer nó da árvore maior.
Profundidade: a profundidade de um nó é o número de ponteiros entre esse nó e a raiz.
Altura: a altura de um nó é o número de ponteiros no caminho mais longo entre um nó e um nó de folha.
Grau: o grau de um nó se refere ao número de subárvores.

<Por que usamos árvores?>

A estrutura hierárquica dá a uma árvore propriedades únicas para armazenar, manipular e acessar dados. As árvores formam uma das organizações mais básicas dos computadores.
Podemos usar uma árvore para os seguintes fins:
- Armazenamento como hierarquia. Ex.: Sistemas de arquivos em um computador.
- Armazenamento de informações que desejamos pesquisar rapidamente. Ex.: as árvores AVL e Red-Black que são projetadas para uma busca rápida.
- As árvores são usadas para herança, analisador XML, aprendizado de máquinas e DNS, entre muitas outras coisas.
- Tipos avançados de árvores, como B-Trees e B+ Trees, podem ser usados para indexação de um banco de dados.
- As árvores são ideais para coisas como redes sociais e jogos de xadrez por computador.
- Uma Spanning Tree pode ser usada para encontrar os caminhos mais curtos em roteadores para networking.

{ Tipos de Árvores

Existem muitos tipos de árvores que podemos usar para organizar os dados de forma diferente dentro de uma estrutura hierárquica. A árvore que usamos depende do problema que estamos tentando resolver. Alguns tipos estão listados a seguir:

- N-ary trees
- Balanced trees
- Binary trees
- Binary Search Trees
- AVL Trees
- Red-Black Trees
- 2-3 Trees
- 2-3-4 Trees

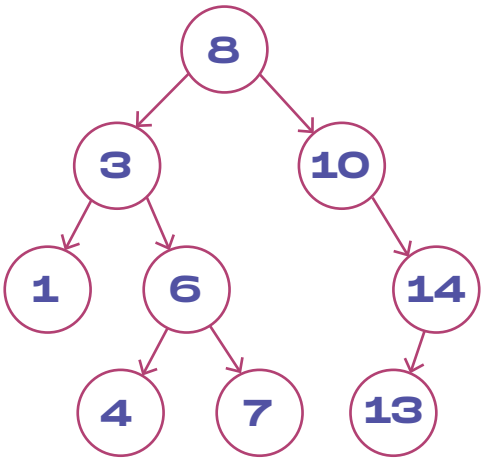
No entanto, as Árvores Binárias, ou Binary Trees, são especiais e muito utilizadas nas mais diversas aplicações porque quando ordenadas permitem que pesquisas, inclusões e exclusões de dados em sua estrutura sejam extremamente rápidas.

<Árvore Binária>

Uma árvore binária é uma estrutura de dados recursiva onde cada nó pode ter no máximo 2 filhos.

Um tipo comum de árvore binária é uma árvore de busca binária, na qual cada nó tem um valor maior ou igual aos valores dos nós na subárvore esquerda e menor ou igual aos valores dos nós na subárvore direita. árvore.

Exemplo de uma Árvore Binária:



Pesquisar em árvores de pesquisa binárias

É importante saber como fazer uma busca em uma árvore. Pesquisar significa que estamos localizando um elemento ou nó específico em nossa estrutura de dados. Como os dados em uma árvore de pesquisa binária são ordenados, a pesquisa é bastante fácil.

1. Comece na raiz.
2. Se o valor for menor que o valor do nó atual, percorremos a subárvore esquerda. Se for mais, percorremos a sub-árvore direita.
3. Continue esse processo até chegar ao nó com esse valor ou chegar a uma folha, o que significa que o valor não existe.

Percursos em árvores

Para usar árvores, podemos percorrê-las verificando cada nó de uma árvore. Se uma árvore for "percorrida", isso significa que todos os nós foram visitados. Existem 3 maneiras de atravessar uma árvore, conforme descrito a seguir:

Ordem de percurso	Primeira visita	Segunda visita	Terceira visita
Em ordem	Visitar sub-árvore esquerda em ordem	Visitar nó raiz	Visitar sub-árvore direita em ordem
Pré-ordem	Visitar nó raiz	Visitar sub-árvore esquerda em pré-ordem	Visitar sub árvore direita em pré-ordem
Pós-ordem	Visitar sub-árvore esquerda em pós-ordem	Visitar sub-árvore direita em pós-ordem	Visitar nó raiz

Esses 3 processos se enquadram em uma das duas categorias: travessia em largura ou travessia em profundidade.

Pesquisa em profundidade (Depth–First Search, DFS)

Em resumo, seguimos um caminho do nó inicial ao nó final e, em seguida, iniciamos outro caminho até que todos os nós sejam visitados.

Os passos para o DFS algoritmo são os seguintes:

1. Escolha um nó. Empurre todos os nós adjacentes em uma pilha.
2. Retire um nó dessa pilha e empurre os nós adjacentes para outra pilha.
3. Repita até que a pilha esteja vazia ou você tenha alcançado seu objetivo. Ao visitar os nós, você deve marcá-los como “visited” antes de prosseguir, ou ficará preso em um loop infinito.

Pesquisa em largura (Breath–First Search, BFS)

Em resumo, continuamos nível por nível para visitar todos os nós em um nível antes de ir para o próximo. O algoritmo BFS é comumente implementado usando filas e requer mais memória do que o algoritmo DFS. É melhor encontrar o caminho mais curto entre dois nós.

Os passos para o BFS algoritmo são os seguintes:

1. Escolha um nó. Enfileire todos os nós adjacentes em uma fila. Retire um nó da fila e marque-o como visitado. Enfileire todos os nós adjacentes em outra fila.
2. Repita até que a fila esteja vazia ou que você tenha atingido seu objetivo.
3. Marcar os nós como “visited” antes de prosseguir.

Leitura complementar:

https://www.baeldung.com/java-binary-tree

https://www.educative.io/blog/data-structures-trees-java

Referências:

https://www.devmedia.com.br/trabalhando-com-arvores-binarias-em-java/25749

https://www.javatpoint.com/binary-tree-java

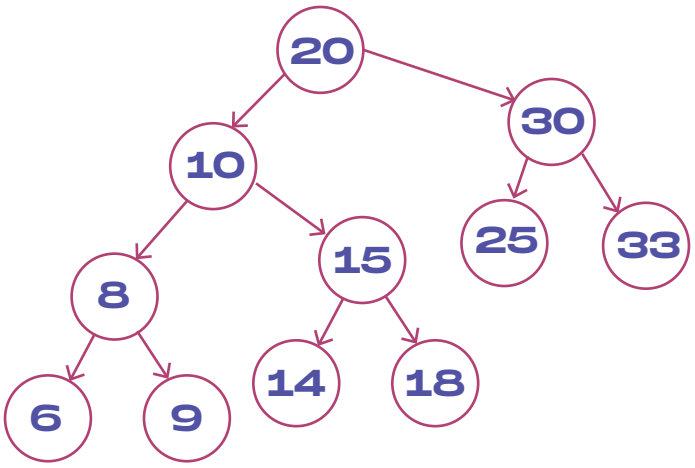
Google olabi

<ÁRVORESEMJAVA>

/exercícios/

1 ➔ Percorrer em profundidade para a árvore binária a seguir:
Percorrer em pré-ordem (pré-fixado);
Percorrer em pós-ordem (pós-fixado);
Percorrer em in-ordem (central).

2 ➔ Percorrer em extensão a árvore binária a seguir, e apresentar o caminho percorrido.



Exercícios 1 e 2

3 ➔ Escreva um algoritmo para calcular a altura de um árvore binária.

4 ➔ Escreva um algoritmo que conte o número de nós de uma árvore binária.

5 ➔ Escreva um algoritmo que conte o número de folhas de uma árvore binária.