Árvore

Estrutura de dados hierárquica que consiste em nós conectados por arestas.

Uma árvore do tipo T é:
- uma estrutura vazia; ou
- um nó do tipo T (chamado raiz) com2 um número finito de árvores do tipo T associadas, chamadas de sub-árvores da raiz.

Conceitos:
- Nó: representa o dado. Cada nó pode ter 0 ou mais nós filhos.

- Nó raiz: é o nó principal da árvore, de onde todos os outros nós descendem. Em uma Representação gráfica, o nó raiz tipicamente fica mais ao topo.
- **Nó folha:** Nó que não tem filho(s).
- Sub-árvore: Qualquer nó, junto com todos os seus descendentes, forma uma sub-árvore.
- Altura da árvore: maior distância entre a raiz e uma folha. É o número máximo de arestas do caminho mais longo da raiz até uma folha. Toda folha tem altura zero. Convenção: h(árvoreVazia) = -1.

- Nó pai e nó filho: Um nó y abaixo de um nó x é chamado filho de x. x é dito pai de y.
- Nó irmão: nós com o mesmo pai são ditos irmãos.
- Nível de um nó: a raiz de uma árvore tem nível 1. Se um nó tem nível i, seus filhos têm nível i+1.

Obs: há autores que dizem que a raíz tem nível 0 (http://www.decom.ufop.br/marco/site_media/uploads/pcc104/09_aula_09.pdf). Aqui vamos assumir que tem nível 1.
- Grau de um nó: quantidade de filhos do nó.

Nas árvores, nós podem ter uma limitação global de filhos. Uma árvore é dita n-ária se cada nó pode ter no máximo n filhos. Um exemplo é a árvore binária, em que cada nó pode ter até dois filhos.

A estrutura típica de uma árvore binária em C é declarada da seguinte forma:

```
struct No {
    int dado;
    struct No *esquerda;
    struct No *direita;
};
```

Uma árvore é dita balanceada ou perfeitamente balanceada se, para todo nó da árvore, os números de nós das suas duas subárvores diferem no máximo em um.

Quando uma árvore precisa cumprir o balanceamento, são realizadas operações chamadas de "rotação", que ajustam a estrutura da árvore para garantir que ela permaneça equilibrada. Essas rotações são realizadas para manter a altura da árvore o mais baixa possível, evitando que a árvore se aproxime de uma lista.

Existem dois tipos principais de rotação:
- **Rotação à esquerda:** move o nó à direita para cima e o nó atual para a esquerda, com o filho esquerdo do novo nó tornando-se o filho direito do nó original.
- **Rotação à direita:** move o nó à esquerda para cima e o nó atual para a direita, com o filho direito do novo nó tornando-se o filho esquerdo do nó original.

Essas rotações são usadas em árvores balanceadas, como as árvores AVL e rubro-negras, para garantir o equilíbrio entre o número de nós em cada subárvore, mantendo a eficiência das operações de busca, inserção e deleção.

Aplicações:
- Sistema de Arquivos: representação de diretórios e arquivos.
- Hierarquias: estruturas organizacionais, como organogramas.
- Compiladores: análise sintática com árvores de derivação.
- Processamento de texto: árvores de sufixo.
- Compressão de dados: geração de árvore (de Huffman) para estabelecer os códigos de cada símbolo a ser compactado.
- Sistemas de decisão: ramificações com caminhos distintos a partir de um caminho atual para tomar uma decisão. Usado, por exemplo, para verificar movimentos possíveis e resultados potenciais em uma partida de xadrez.

Árvores Binárias de Busca (BST)

Uma Árvore Binária de Busca (Binary Search Tree - BST) é uma árvore binária em que, para cada nó, todos os nós da sub-árvore:
- esquerda têm valores menores que o nó em questão.
- direita têm valores maiores que o nó em questão.

Numa árvore binária de busca com n chaves e de altura h, as operações de busca, inserção e remoção têm complexidade de tempo O(h). Se a árvore estiver desbalanceada, sua altura pode chegar a n, assim, operações de busca, inserção e remoção podem chegar a O(n) no pior caso.

- BST.c

Ver arquivo Arvore.hs.

Percurso em árvores binárias:

Pré-ordem: Raiz -> Esquerda -> Direita, ou seja:
- Visitar a raiz.
- Percorrer a sua subárvore esquerda em pré-ordem.
- Percorrer a sua subárvore direita em pré-ordem.
- Aula-2-Figura-3-Pré-ordem.png

Em-ordem: Esquerda -> Raiz -> Direita, ou seja:
- Percorrer a sua subárvore esquerda em in-ordem.
- Visitar a raiz.
- Percorrer a sua subárvore direita em in-ordem.
- Aula-2-Figura-4-Em-ordem.png

Pós-ordem: Esquerda -> Direita -> Raiz, ou seja:
- Percorrer a sua subárvore esquerda em pós-ordem.
- Percorrer a sua subárvore direita em pós-ordem.
- Visitar a raiz.
- Aula-5-Figura-5-Pós-ordem.png

Em Largura: Raiz -> Filho à esquerda -> Filho à direita, ou seja:
- Visitar a raiz.
- Visitar a subárvore esquerda.
- Visitar a subárvore direita.
- Aula-5-Figura-6-Em-largura.png

Observar arquivos:
- percurso.arvores.c
- percurso.arvores.cpp
- percurso.arvores.py

Árvores AVL

- Árvores AVL (Georgy Adelson-Velsky & Evgenii Landis) são uma variação das Árvores Binárias de Busca que mantêm a árvore balanceada. É uma das primeiras estruturas de dados de árvores autobalanceadas.
- Busca, inserção e remoção com complexidade O(log n).

Árvores rubro-negras

- Variação das árvores binárias de busca autobalanceadas.
- Cada nó da árvore possui um atributo "cor" (vermelho ou preto).
- A árvore mantém regras específicas para garantir que o caminho mais longo da raiz a uma folha seja no máximo duas vezes o caminho mais curto. Isso garante que a árvore permaneça aproximadamente balanceada.

- Os nós folha são sempre pretos.
- A raiz é preta.
- Não há dois nós vermelhos adjacentes.
- Busca, inserção e remoção com complexidade O(log n).

As árvores rubro-negras são bastante utilizadas. Exemplos:
- postgres: implementação de índices.
- Java: java.util.TreeMap.
- C++ STL: std::map e std::set.
- kernel Linux: escalonador.

Na prática, a árvore AVL é mais rápida na operação de busca, e mais lenta nas operações de inserção e remoção.
A árvore AVL é mais balanceada do que a árvore Rubro-Negra, o que acelera a operação de busca.

Simuladores:
- BST: https://www.cs.usfca.edu/~galles/visualization/BST.html
- AVL: https://www.inf.ufsc.br/~aldo.vw/estruturas/simulador/AVL.html
- Rubro-negras:
https://www.inf.ufsc.br/~aldo.vw/estruturas/simulador/RB.html