Árvores

1. Introdução

Até agora vimos estruturas de dados lineares, especialmente a lista encadeada e suas variações (ordenada, duplamente encadeada, circular, com nodo cabeça) e casos particulares (fila e pilha). Nas listas lineares existe um único caminho a ser percorrido do primeiro elemento até o último, e para percorrer a lista, partindo do primeiro elemento é preciso passar por todos os outros elementos até chegar ao último.

No entanto, é possível organizar outros tipos de estruturas de dados com diferentes formas de conexão, com múltiplos caminhos, permitindo mais de um percurso possível. Lembrando que percorrer uma estrutura de dados significa passar por todos os elementos uma única vez. Ou seja, nenhum elemento deixa de ser visitado, e nenhum elemento é visitado mais de uma vez.

Uma estrutura de dados não linear clássica é a árvore.

2. Definições

Uma árvore é uma estrutura de dados onde os nodos estão organizados de forma hierárquica. O primeiro nodo é chamado de nodo **raiz**. A esse nodo raiz podem estar ligados outros nodos chamados de **ramos** ou **filhos**. Cada um desses nodos filhos é também uma árvore, e pode ter outros nodos filhos associados a eles. Com exceção da raiz, **cada nodo** é associado somente a um nodo pai, e **pode ter de 0 a *n* nodos filhos**, para uma **árvore de ordem *n***, de modo que só exista um único caminho entre a raiz e qualquer outro nodo.

O número de filhos que um nodo possui é chamado de grau do nodo.

Os nodos que não tem filhos são chamados de **folhas**.

Exemplo:

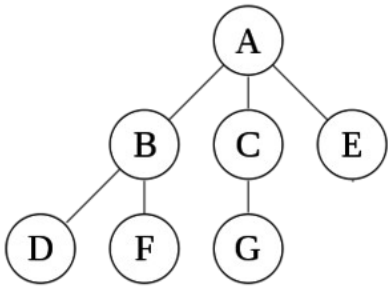

Figura 1. Árvore de ordem 3 com nodos A, B, C, D, E, F e G.

No exemplo da figura 1, o nodo A é a raiz da árvore, e possui três filhos, B, C e E. Os nodos D, E, F e G são nodos folha (não tem filhos). Observe que há um único caminho possível a partir da raiz para chegar a qualquer nodo.

A **profundidade** de um nó é a distância deste nó até a raiz. Um conjunto de nós com a mesma profundidade é denominado **nível da árvore**. A maior profundidade de um nó, é a **altura da árvore**.

Uma árvore é um tipo específico de um grafo. Um grafo é uma estrutura de dados que pode representar uma rede de nós. A árvore é um tipo específico de grafo sem ciclos, ou seja, com um único caminho entre dois nodos quaisquer.

Dois exemplos de usos práticos da estutura de árvore são a árvore genealógica e a árvore de diretórios do sistema operacional.

Para nossa disciplina de estruturas de dados nosso interesse é voltado para as árvores de pesquisa.

3. Árvore Binária

Inicialmente veremos a árvore de pesquisa binária, que é uma árvore de ordem 2. Ou seja, cada nodo pode conter de 0 a 2 filhos.

Uma árvore binária é uma estrutura de dados caracterizada por:

- Ou não tem elemento algum (árvore vazia).
- Ou tem um elemento distinto, denominado raiz, com dois ponteiros para duas estruturas diferentes, denominadas subárvore da esquerda e subárvore da direita.

Perceba que a definição é recursiva e, devido a isso, muitas operações sobre árvores binárias utilizam recursão. É o tipo de árvore mais utilizado na computação. A principal utilização de árvores binárias são as árvores binárias de busca.

Os dados são inseridos na árvore respeitando uma ordem. O mais comum é os elemento menores que o elemento da raiz são inseridos na subárvore da esquerda e os elementos maiores que a raiz são inseridos na subárvore da direita.

3.1. Inserir elemento na árvore binária.

No caso de inserir um elemento menor que a raiz, se a subárvore da esquerda estiver vazia, é criado um novo nodo como filho da esquerda da raiz. O mesmo se aplica se o elemento for maior que a raiz, e a subárvore da direita estiver vazia.

Por exemplo, ao inserir a sequência 15, 10, 12, 9, 5, 33, 21, 25 e 13 em uma árvore a vazia, o resultado é o representado na figura 2.

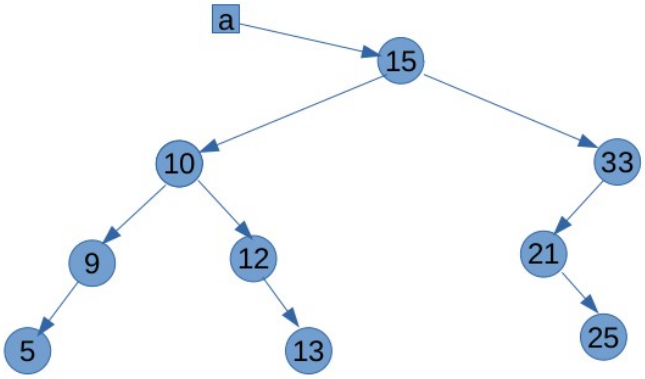

Figura 2. Resultado da inserção dos elementos `15`, `10`, `12`, `9`, `5`, `33`, `21`, `25` e `13` na árvore binária vazia `a`.

Observe que a disposição da árvore depende da ordem da inserção. Se inserirmos os mesmos elementos, em outra ordem, a disposição da árvore será diferente. Por exemplo, a figura 3

representa o resultado da inserção dos mesmos elementos, mas na ordem inversa (13, 25, 21, 33, 5, 9, 12, 10 e 15).

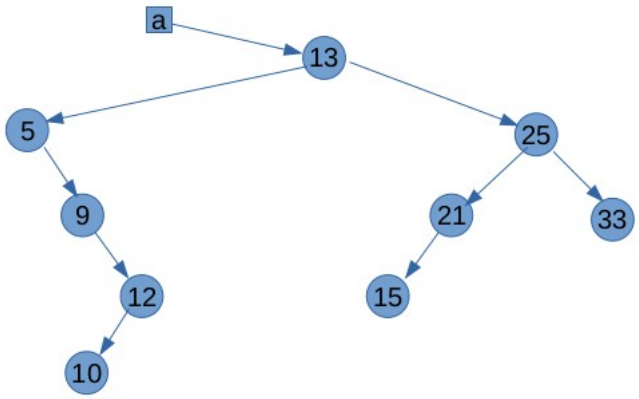

Figura 3. Resultado da inserção da sequência 13, 25, 21, 33, 5, 9, 12, 10 e 15 na árvore vazia a

Observe que as estruturas das árvores das figuras 2 e 3 ficaram bem diferentes apenas invertendo a ordem de inserção.

A árvore da figura 2 tem altura 4 e a árvore da figura 3 tem altura 5.

3.2. Remover elemento da árvore binária.

Para remover elementos da árvore binária é preciso localizar o elemento, e refazer as conexões da árvore. Mas agora não há somente um prosseguimento da árvore. Então é preciso tomar uma decisão de qual elemento colocar no lugar do elemento removido, de modo a manter a integridade da árvore, ou seja, manter a mesma regra de formação, que diz que os elementos menores que a raiz são alocados na subárvore da esquerda e os elementos maiores que a raiz são alocados na subárvore da direita.
Dessa forma, a remoção precisa considerar os possíveis casos do nodo que será removido.
1. Nodo sem filhos (nodo folha). Nesse caso, basta remover o próprio nodo. Por exemplo, para remover o nodo 5, 13 ou 25 da figura 2, basta remover o próprio nodo.

2. Nodo com somente um filho. Nesse caso, o filho do nodo removido passa a ser conectado diretamente ao pai do nodo removido. Por exemplo, para remover o nodo 9 da figura 2, basta fazer o nodo 5 se tornar o filho da esquerda do nodo 10.

3. Nodo com dois filhos. Nesse caso, pode-se optar por colocar no lugar do nodo removido o maior elemento da subárvore da esquerda ou o menor elemento da subárvore da direita. Isso porque esses dois elementos são os que estão numericamente mais próximos do valor do elemento a ser removido. Na codificação o programador escolhe um dos lados para remover. Por exemplo, para remover o elemento 15 (raiz) da árvore da figura 2, o elemento que deve ser movido para a raiz podemos colocar no lugar do 15 o elemento 13 (maior elemento da subárvore da esquerda) Observe que o elemento 13 na raiz consegue manter a estrutura da árvore, pois todos os elementos da esquerda da raiz são menores que o 13 e os elementos à direita da raiz são maiores que o 13. A outra opção possível é o elemento 21 (menor elemento da subárvore da direita). Se fizermos essa escolha, o elemento 21 também consegue manter a estrutura da árvore, pois todos os elementos à esquerda da raiz são menores que o 21 e os da direita são maiores que o 21.

3.4. Percursos.

Diferentemente de uma lista encadeada, que é uma estrutura linear e só possui uma forma de percorrê-la, a árvore não é linear, e existem 3 formas clássicas de percorrer toda a árvore, passando por todos os elementos processando cada elemento uma única vez.

3.4.1. Percurso em pré-ordem

No percurso em pré-ordem a raiz é visitada (processada) antes de visitar os filhos.
Nesse percurso, primeiro a raiz é visitada, e depois, chama-se recursivamente o percurso em pré-ordem para a subárvore da esquerda e para a subárvore da direita.

A sequência dos elementos pesquisados em pré-ordem é:
Para o exemplo da figura 2: 15, 10, 9, 5, 12, 13, 33, 21, 25.
Para o exemplo da figura 3: 5, 9, 12, 10, 21, 15, 33, 25, 13.

Utilidade. O percurso em pré-ordem produz uma sequência de elementos que podem gerar a mesma estrutura da árvore original (backup).

3.4.2. Percurso em in-ordem

No percurso em in-ordem, a raiz é visitada depois de percorrer em in-ordem toda a subárvore da esquerda, e antes de percorrer em in-ordem toda a subárvore da direita.

A sequência da impressão dos elementos em in-ordem é:
Para o exemplo da figura 2: 5, 9, 10, 12, 13, 15, 21, 25, 33.
Para o exemplo da figura 3: 5, 9, 10, 12, 13, 15, 21, 25, 33.

Utilidade. Observe que o percurso in-ordem visita os nodos em ordem crescente.
A lógica é a seguinte. Imprime primeiro a subárvore da esquerda (cujos elementos são menores que a raiz), depois imprime a raiz, e depois imprime a subárvore da direita (cujos elementos são maiores que a raiz).

3.4.3. Percurso em pós-ordem

No percurso em pós-ordem, a raiz é visitada depois de percorrer em pós-ordem toda a subárvore da esquerda e a subárvore da direita.

A sequência de impressão dos elementos em pós-ordem é:
Para o exemplo da figura 2: 5, 9, 13, 12, 10, 25, 21, 33, 15.
Para o exemplo da figura 3: 10, 12, 9, 5, 15, 21, 33, 25, 13.

Utilidade. O percurso em pós-ordem serve para remover todos os elementos da árvore.

3.5 Busca de um elemento na árvore.

A busca de um elemento da árvore é muito mais rápida do que a busca em uma estrutura linear, onde é preciso passar por todos os elementos para encontrar o último elemento da lista.
Na árvore binária de pesquisa, nem todos os elementos são pesquisados durante a busca. Isso ocorre porque há uma regra de composição da árvore.
Se o elemento que estou buscando é menor que a raiz, ele só pode estar na subárvore da esquerda. Então, basta buscar na subárvore da esquerda. Se não encontrar na subárvore da esquerda, o elemento não está na árvore.

O mesmo vale se o elemento for maior que a raiz, buscando na subárvore da direita.
Se o elemento for igual ao elemento da raiz, ele foi encontrado.

Por exemplo, para buscar os elementos 4 (inexistente), e 21 (existente), nos dois exemplos de árvores).
Para buscar o elemento 4 na árvore da figura 2, são comparados os elementos 15, 10, 9 e 5. Como o elemento 5 não tem filho da esquerda, e o elemento buscado 4 é menor que o 5, ele só poderia estar na subárvore da esquerda do 5, e como o 5 não tem filhos, a conclusão é que o 4 não está na árvore. Foram no total 4 comparações.

Para buscar o elemento 21 na árvore da figura 2, são comparados os elementos 15, 33 e 21. Ao chegar no 21, o elemento é encontrado. Foram no total 3 comparações.

Para buscar o elemento 4 na árvore da figura 3, são comparados os elementos 13 e 5. Como o elemento 5 não tem filho da esquerda, e o elemento buscado 4 é menor que o 5, ele só poderia estar na subárvore da esquerda do 5, e como o 5 não tem filhos, a conclusão é que o 4 não está na árvore. Foram no total 2 comparações.

Para buscar o elemento 21 na árvore da figura 3, são comparados os elementos 13, 25 e 21. Ao chegar no 21, o elemento é encontrado. Foram no total 3 comparações.

Observe que o **pior caso possível**, ou seja, o mais trabalhoso para buscar um elemento em uma árvore binária é fazer o número de comparações igual à **altura da árvore**.