ÀRVORES B

Árvores B: principais definições

Como visto, as árvores binárias armazenam em um nó um único registro e possuem até dois nós (esquerdo e direito) como filhos. Além disso, vimos que a árvore binária de busca tem como característica armazenar na subárvore esquerda de um nó apenas elementos menores do que ele e, na direita, elementos maiores.

Podemos ampliar o conceito de armazenamento ordenado da árvore binária de busca para maiores ordens, ou seja, uma árvore de busca com mais de dois filhos por nó.

Outra característica que pode ser adicionada é armazenar mais de um elemento (registro) por nó. Observe a seguir:

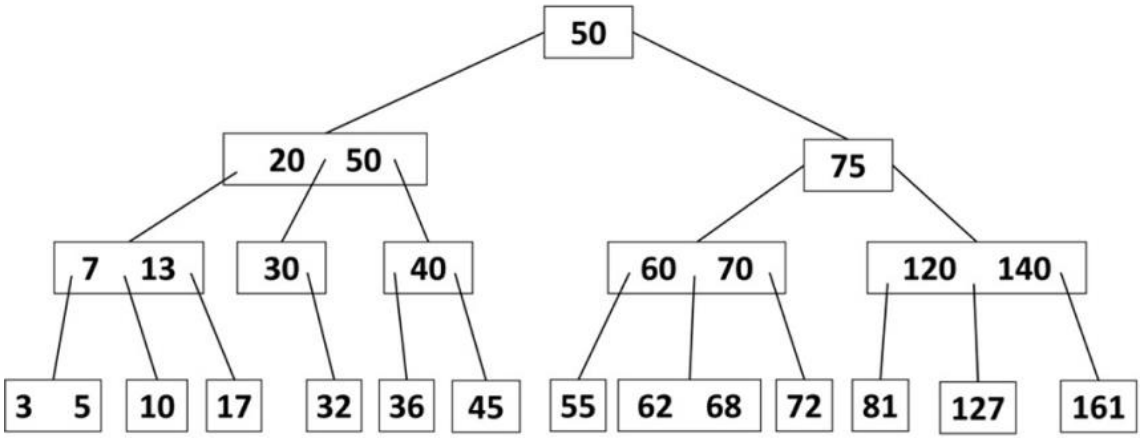

Imagem 28 - A árvore B é um exemplo de árvore de busca de dados multidirecional.

O conceito é importante porque amplia as possibilidades de armazenamento além de manter uma árvore com profundidade menor do que uma árvore binária de busca ou árvore AVL.

"[...] utilizando o recurso de manter mais de uma chave em cada nó da estrutura, proporciona uma organização [...] tal que as operações mencionadas (busca, inserção e remoção) são executadas rapidamente.

As árvores com essas novas características citadas são conhecidas como árvore de busca multidirecional e como árvores B. A árvore B foi proposta por **Rudolf Bayer**.

Não se sabe de onde vem o B do nome da árvore. Uma das hipóteses é que a letra B é a primeira letra do seu sobrenome, Bayer.

As árvores B são muito utilizadas como forma de armazenamento em memória secundária (como HDs e outros dispositivos de armazenamento secundário com acesso direto). As árvores B se destinam a armazenar grandes tabelas, por isso diversos sistemas comerciais de bancos de dados, por exemplo, utilizam esse tipo de árvore (SZWARCFITER; MARKENZON, 1994).

A árvore B é uma árvore balanceada em que o número de nós acessados nas operações de busca/inserção/remoção é muito pequeno se comparado às outras árvores já estudadas.

A árvore B garante que as folhas se encontrem todas em um mesmo nível, independentemente da ordem em que os dados serão inseridos.

Em outras palavras, Preiss (2000) define uma árvore B de ordem como uma árvore de busca com as seguintes propriedades:

- A raiz tem no mínimo duas e no máximo **n** subárvores.
- Cada um dos nós internos (diferentes da raiz) tem entre ⌈**n**/2⌉ e **n** subárvores e entre ⌈**n**/2⌉−1 e **n**−1 elementos.
- Todos os nós folhas estão no mesmo nível.

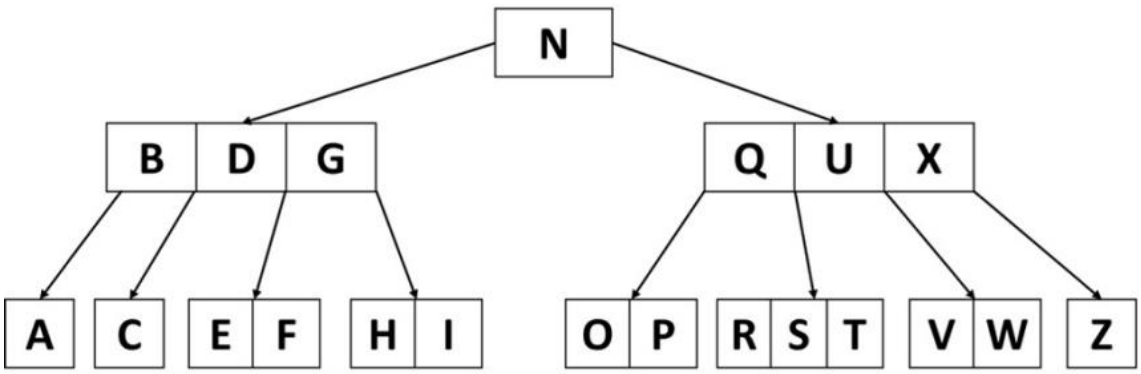

Imagem 29 - Árvore B.

Na imagem 29, temos um exemplo de árvore B de ordem 4. Observe que essa árvore obedece à definição:

- A raiz possui duas subárvores (quantidade mínima).
- Todas as folhas estão em um mesmo nível.
- Todos os nós internos (que não são folhas e são diferentes da raiz) possuem quatro subárvores, ou seja, entre ⌈**n**/2⌉ e **n**.

Sabemos que em uma árvore de ordem **n** (em que **n** é um número natural maior do que um), a árvore B pode ter **n** filhos. Além disso, é balanceada. Nesse caso, precisamos ter algoritmos que garantam essa característica à árvore. Um elemento inserido em uma árvore B é sempre colocado em uma folha. Assim como nas outras árvores e, de acordo com a necessidade, as redistribuições são feitas a fim de garantir que a árvore fique sempre balanceada.

A árvore B generaliza as árvores binárias de busca, mais especificamente, a árvore AVL.

Árvores B: busca

A busca realizada em uma árvore B é semelhante à busca em uma árvore AVL e, consequentemente, a de uma árvore binária de busca. Como é possível armazenar mais de um elemento em um nó, é necessário adicionar testes a serem realizados em cada nó para indicar qual o próximo passo.

Na imagem 30, temos um exemplo de busca em árvore B. Considere a busca da chave 75. A primeira comparação a ser feita é com a raiz. Como o elemento 75 é maior que o 60, está à direita de 60, segue a busca para o nó que armazena os elementos 72 e 81. Nesse nó, são realizadas novas comparações a fim de determinarmos para qual filho a busca deverá seguir. Como o elemento é maior do que 72 e menor do que 81, a busca segue para o filho localizado entre 72 e 81. O caminho percorrido nessa busca está destacado na imagem 30:

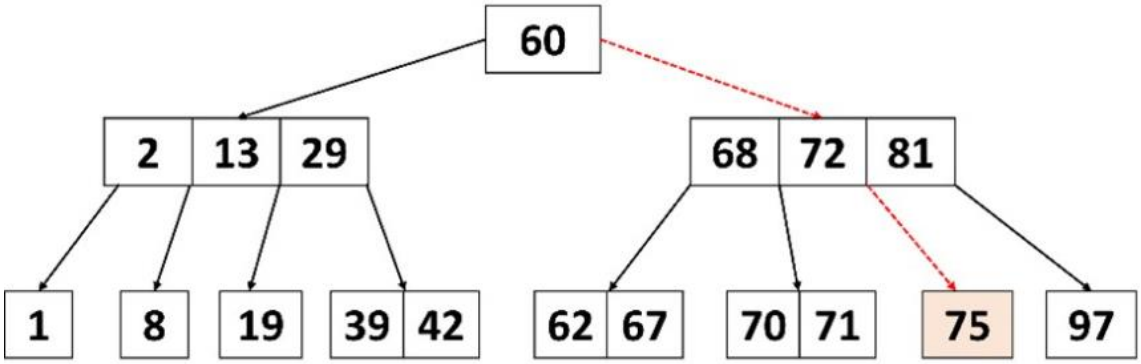

Imagem 30 - Exemplo de busca em uma árvore B.

A análise de complexidade está relacionada ao número de acesso aos nós (**O**(log **n**)) e pela busca linear em cada nó considerado na busca (**T** em cada nó). Portanto, **O**(**T** log **n**).

Árvores B: inserção

Na inserção de chaves na árvore B, devemos nos preocupar com a definição quanto à ordem da árvore, porque um nó interno (que é diferente da raiz) deve possuir entre [**n**/2] e **n** subárvores.

1. O primeiro passo da inserção é buscarmos a folha na qual será inserido o novo elemento.
2. No segundo passo, devemos verificar se a folha está ou não completa e, se não estiver completa, basta incluir o elemento em sua posição correta dentro do nó.
3. Se o nó estiver cheio, devemos proceder ao segundo passo de forma diferente.
    1. Se o nó estiver cheio, ao invés de criar um nó com apenas esse novo elemento, devemos dividir o nó cheio. Assim, se algum nó ficar com ordem maior do que n, o nó deve ser dividido. Para isso, o valor central do nó que será dividido sobe para a posição do pai, repetindo recursivamente por toda árvore até a raiz (se necessário).
4. Se acontecer da raiz estourar, um novo nó raiz será criado e poderá ter somente um elemento, se necessário.

É importante acrescentar que, por proceder à inserção da forma descrita anteriormente, é que se garante que todas as folhas sempre estarão em um mesmo nível. Vamos acompanhar esses passos de forma figurada.

Seja a árvore B retratada na imagem a seguir:

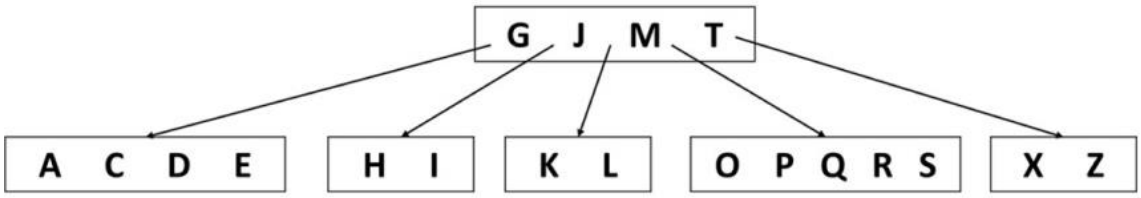

Imagem 31 - Árvore B para inserção.

A partir da árvore B inicial da imagem 31, devemos proceder com as inserções dos seguintes elementos, respectivamente: B, N, F. A inserção do elemento B resulta na árvore da imagem 32:

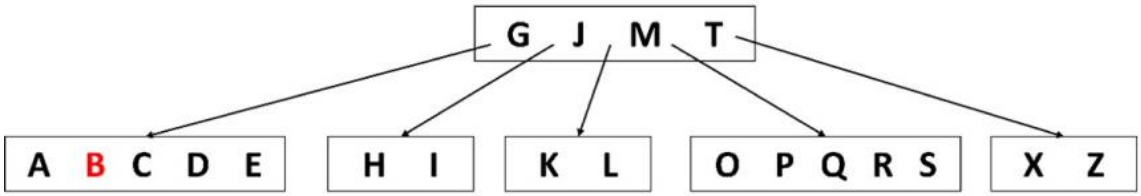

Imagem 32 - Inserção do elemento B na árvore da imagem 31.

Na sequência, deve ser inserido o elemento N. Contudo, o nó em que deve ser inserido N já está cheio. Assim, devemos dividir o nó em dois e subir em um nível o elemento central. Lembre-se de que esse processo repete recursivamente até a raiz. A árvore resultante após a inserção de N é apresentada na imagem 33:

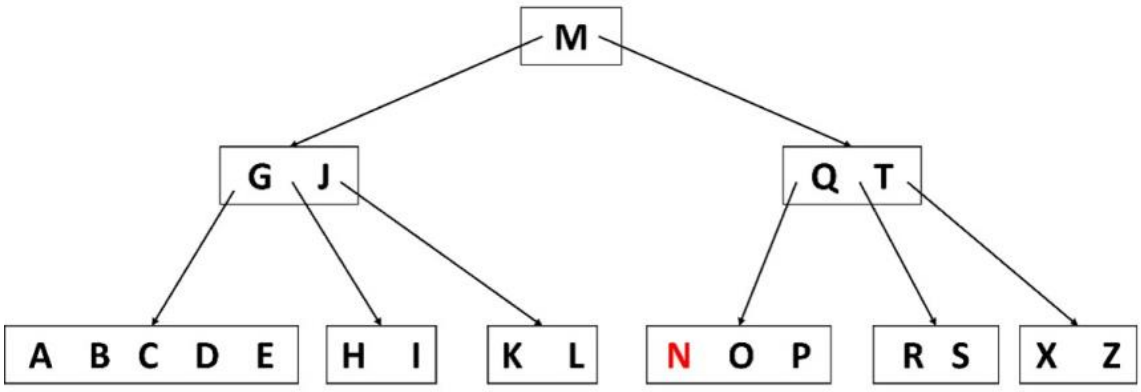

Imagem 33 - Inserção do elemento N na árvore B da imagem 32. Observe que, nesse passo, há necessidade de subida de níveis para “comportar” o novo elemento.

O último elemento a ser inserido é o F. Mais uma vez, o nó em que deveria ser inserido o novo elemento, nesse caso F, já está cheio. Procedemos à divisão do nó em dois e subimos o elemento central. A imagem a seguir apresenta o resultado final das inserções da árvore B:

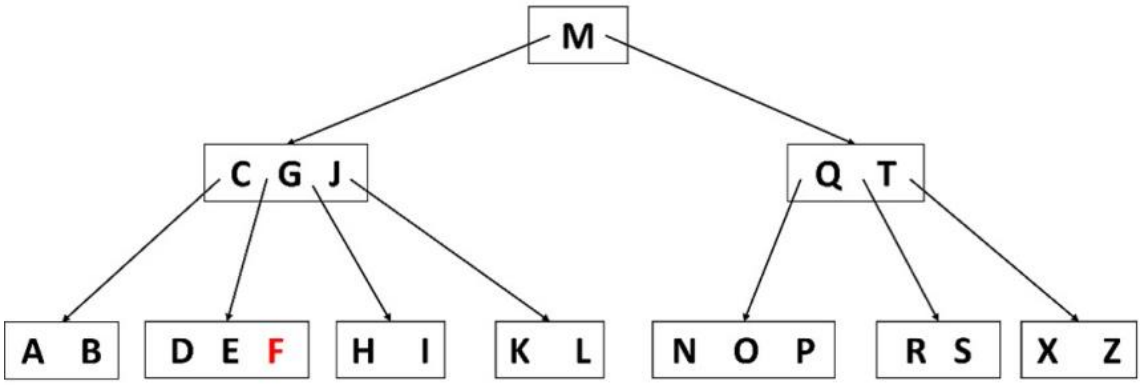

Imagem 34 - Inserção do elemento F na árvore B da imagem 32. Novamente, há necessidade de subida de níveis para “comportar” o novo elemento.

A análise de complexidade da inserção na árvore B está relacionada ao número de acesso aos nós (**O**(log **n**)) e pela busca linear em cada nó (**t** em cada nó) para encontrar o local a ser inserido o elemento. Portanto, **O**(**t** log **n**).

Árvores B: remoção

Para a remoção de um elemento na árvore B, existem dois casos que devem ser observados:

- O elemento a ser inserido estar ou não em uma folha.
- Ao remover um elemento, um nó não atingir a quantidade mínima de elementos que devem estar armazenados no nó, de acordo com a ordem da árvore.

O algoritmo de remoção ficará da seguinte forma:

1. Aplicar o procedimento de busca, verificando a existência do elemento a ser removido na árvore. 1
2. Se o elemento a ser removido se encontrar em uma folha, o elemento é simplesmente retirado.

3. Se o elemento a ser removido não se encontrar em uma folha (localizar-se em um nó interno), é substituído pelo elemento imediatamente maior ou imediatamente menor. Observe que o elemento imediatamente maior ou menor pertence, por definição, a uma folha. Com isso, reduzimos para o caso de remover o elemento da folha.

4. Se o nó continuar com o número mínimo de elementos, fim.

5. Se não, a folha obrigatoriamente tem uma chave a menos que o mínimo. Assim, verifique os nós irmãos (imediatamente à esquerda e direita):

    1. Se um dos nós irmãos tiver mais do que o número mínimo de elementos, aplique redistribuição. Lembre-se de que um nó interno deve possuir um mínimo de n/2 - 1 elementos. A redistribuição consiste em concatenar o nó folha do elemento removido com o do irmão adjacente, o que resulta em um nó maior do que o aceito pela definição. Em seguida, efetue a divisão do nó em dois e suba em um nível o elemento central, igual à divisão da inserção.
    2. Se não, concatene o nó com um dos irmãos e o elemento separador do pai.

6. Se ocorrer concatenação, volte ao passo 4 com o nó pai, porque remover um elemento do pai pode acarretar de o pai não ter mais o número mínimo de elementos.

Proceder à remoção da forma descrita anteriormente garante que todas as folhas sempre estarão em um mesmo nível. Vamos acompanhar esses passos de forma figurada.

Veja a árvore B retratada na imagem 35:

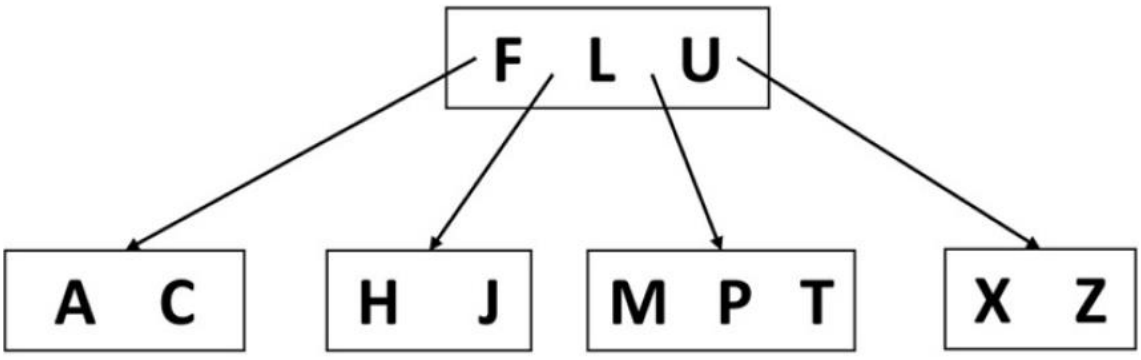

Imagem 35 - Árvore B para remoção.

A partir da árvore B inicial da imagem 35, devemos proceder às remoções dos seguintes elementos respectivamente: J e L.

- No passo 1 da remoção, deve-se buscar o elemento a ser removido da árvore. É identificado o nó que contém os elementos H e J.
- Como o elemento se encontra em uma folha, remova o elemento (passo 2).
- Já que o nó não continua com o número mínimo de elementos (passo 5), deve-se executar o passo 5.1 (redistribuição).
- Faça a redistribuição com o nó irmão da direita. Após isso, a remoção é realizada.

O resultado da remoção é apresentado na imagem a seguir:

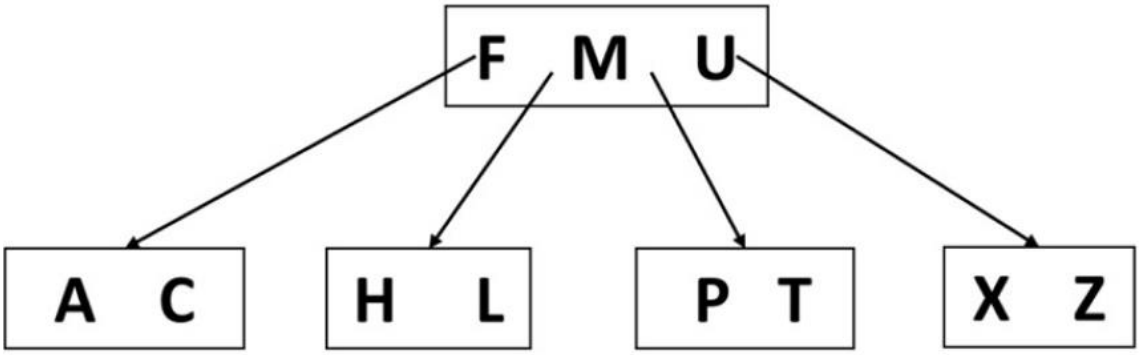

Imagem 36 - Resultado da remoção do elemento J. Observe que houve um reposicionamento dos elementos na raiz.

Vamos remover o elemento L:

- Como passo 1 da remoção, deve-se buscar o elemento a ser removido da árvore. Assim, é identificado o nó que contém os elementos H e L.
- Como o elemento se encontra em uma folha, remova o elemento (passo 2).
- Uma vez que o nó não continua com o número mínimo de elementos (passo 5), execute o passo 5.2 (concatenação).
- A concatenação é feita com o pai e o irmão direito, o que resulta em um nó com os elementos H, M, P, T.

- Execute o passo 6 voltando ao passo 4 para o pai (recursão, nó com os elementos F, U). Como o nó pai continua obedecendo à definição do número mínimo de elementos, a remoção é finalizada.

O resultado é apresentado na imagem 37:

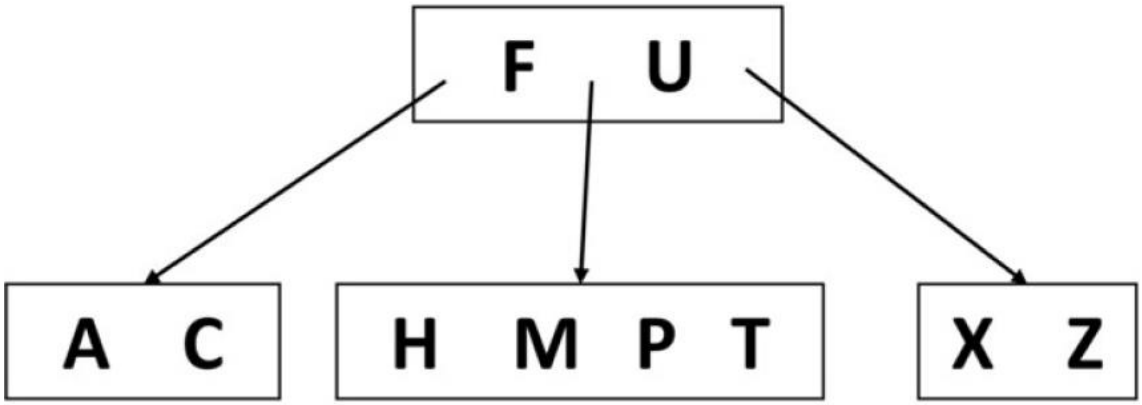

Imagem 37 - Resultado da remoção do elemento L. Observe que houve uma concatenação dos elementos de dois (2) nós folhas.

Cada passo relatado na remoção de um elemento possui uma complexidade envolvida. Porém, quase sempre é concentrada na situação de pior caso. A remoção na árvore B também está relacionada ao número de acesso aos nós (**O**(log **n**)) e pela busca linear em cada nó (**t** em cada nó) para encontrar o local a ser inserido o elemento. Portanto, **O**(**t** log **n**).