INF0208 – Estruturas de Dados
Profa.: Helena Graziottin Ribeiro

ÁRVORES BALANCEADAS

O tempo de acesso a um nó, em uma estrutura sob a forma de árvore, é proporcional à quantidade de comparações realizadas até encontrá-lo. Quanto menor este tempo, maior é a eficiência no seu acesso. Uma árvore que mantém seus nós de forma ordenada, como a árvore binária de pesquisa, permite otimizar os acessos, uma vez que a busca é direcionada pela ordenação do nó pesquisado em relação aos demais (> ou <), evitando a procura na árvore inteira. A eficiência no acesso a um nó está relacionada a dois fatores:

- freqüência de acesso a cada nó;
- organização da árvore.

A **freqüência de acesso** é determinada pela quantidade de vezes que um nó é procurado. Assim, se todos os símbolos forem igualmente procurados essa freqüência é uniforme. Quanto à **organização da árvore**, pode-se verificar que a quantidade de comparações necessárias numa árvore ordenada para localizar um nó está relacionada com a distância entre este nó e a raiz da árvore. Assim, quanto mais perto da raiz estiverem os nós da árvore, mais depressa eles podem ser encontrados. Se a distância de cada um dos nós em relação à raiz for igual ou bastante próxima à *média das distâncias dos nós em relação à raiz*, o tempo de acesso a cada um deles será próximo aos dos demais. Já se uma sub-árvore A_1 da árvore A tiver muitos níveis e outra sub-árvore A_2 poucos níveis, procurar um nó em A_1 poderá levar muito mais tempo do que procurar um outro nó em A_2, principalmente se em ambos os casos os nós procurados forem folhas.

Uma árvore que esteja organizada de forma que, para qualquer nó, o comprimento da sua sub-árvore mais à esquerda seja igual ou com uma diferença mínima em relação aos comprimentos das suas demais sub-árvores, é denominada de **árvore balanceada**. Há diferentes propostas de estruturação de árvores balanceadas, como Árvores AVL, Árvores 2-3, Árvores-B e Árvores B+.

No caso das árvores binárias, a ordem na qual os símbolos são instalados na árvore é que vai determinar a eficiência da localização dos mesmos. E não é possível pré-determinar a ordem em que eles serão inseridos, para que se possa obter uma árvore balanceada. Dois casos bastante desfavoráveis são:

a) símbolos instalados na ordem de classificação (exemplo: 2-3-4-5-6-7-8-9);
b) símbolos instalados em ziguezague: o primeiro, o último, o segundo, o penúltimo, o terceiro, o antepenúltimo, etc. (exemplo: 2-9-3-8-4-7-5). Estas situações geram **árvores degeneradas**, nas quais o tempo de acesso a cada elemento seria o mesmo para uma representação por listas.

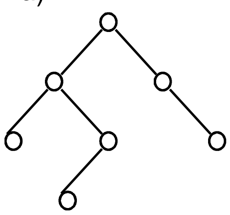
Árvores AVL (AVL-Trees)

As árvores binárias de pesquisa têm uma séria desvantagem que pode afetar o tempo necessário para recuperar um item armazenado, já que a estrutura da árvore ***depende da ordem em que os elementos são inseridos***. Se os elementos forem inseridos já em ordem, a estrutura da árvore será igual a de uma lista encadeada e o tempo médio para recuperar uma informação da árvore aumenta.

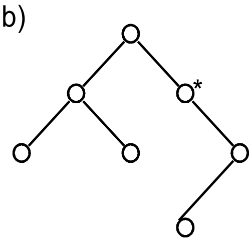
Uma maneira de corrigirmos esta deficiência das ABP é controlar a montagem da estrutura da árvore. Idealmente queremos que a árvore esteja **balanceada**, ou seja, para um nodo **p** qualquer a altura da subárvore esquerda é **aproximadamente** igual à altura da subárvore direita. Obviamente há um custo extra de processamento para manter a árvore balanceada, mas que é compensado quando os dados armazenados precisam ser recuperados muitas vezes.

A idéia de manter uma árvore binária balanceada dinamicamente, ou seja, enquanto os nodos estão sendo inseridos foi proposta em 1962 por 2 soviéticos chamados **A**delson-**V**elskii e **L**andis. Este tipo de árvore ficou então conhecida como **árvore AVL**, pelas iniciais dos nomes dos seus inventores. Pertencem ao grupo das árvores balanceadas pela altura (*height-balanced trees*). São árvores binárias de pesquisa nas quais, para qualquer nó, o comprimento de sua sub-árvore da esquerda não pode ser diferente do comprimento de sua sub-árvore da direita em mais de um nó (ou seja, a diferença é no máximo de 1 nó). Nos exemplos abaixo, (a) e (c) são árvores balanceadas, e (b) não, por causa do nó marcado com o asterisco.

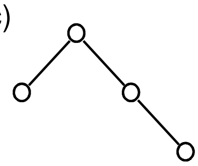
a)



b)



c)

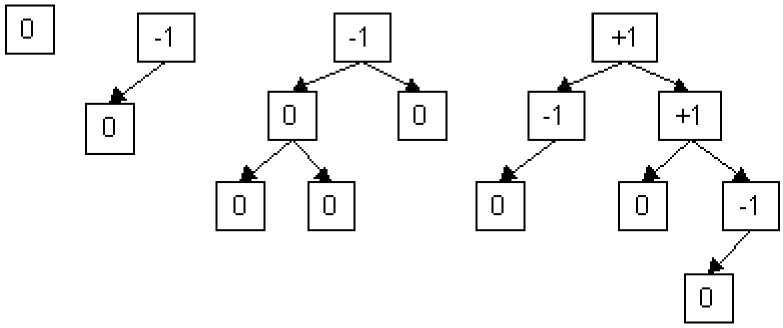


Após a INSERÇÃO ou REMOÇÃO de 1 nó, a árvore poderá ficar desbalanceada (por um nível). Por isso, sempre que uma destas operações for realizada, será necessário verificar o balanceamento. Se a árvore estiver desbalanceada, será necessário aplicar uma função de reestruturação da árvore, a qual, se for preciso, fará uma relocação de todos os nós para que a árvore permaneça ordenada e balanceada.

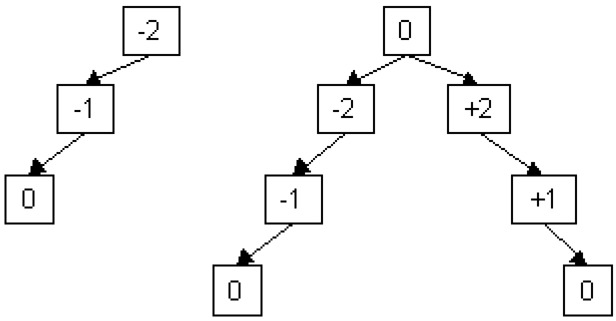
Por definição uma árvore AVL é uma árvore binária de pesquisa onde a diferença em altura entre as subárvores esquerda e direita é no máximo 1 (positivo ou negativo). Assim, para cada nodo podemos definir um **fator de balanceamento (FB)**, que vem a ser um número inteiro igual a

```
FB(nodo p) = altura(subárvore direita p) - altura(subárvore esquerda p)
```

A seguir exemplos de árvores AVL e árvores não-AVL. Os números nos nodos representam o FB para cada nodo. Para uma árvore ser AVL os fatores de balanceamento devem ser necessariamente -1, 0, ou 1.



Exemplos de Árvores AVL

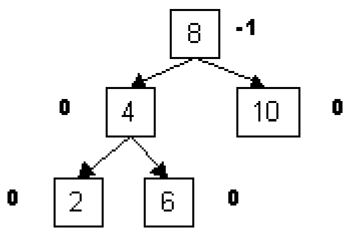


Exemplos de Árvores NÃO-AVL

O balanceamento de uma árvore envolve basicamente operações de: *rotação direita, rotação esquerda, rotação dupla direita e rotação dupla esquerda* [5].

BALANCEAMENTO DE ÁRVORES AVL

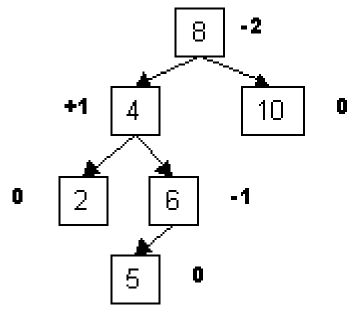
Como fazemos então para manter uma árvore AVL balanceada? Inicialmente inserimos um novo nodo na árvore normalmente. A inserção deste novo nodo pode ou não violar a propriedade de balanceamento. Caso a inserção do novo nodo não viole a propriedade de balanceamento podemos então continuar inserindo novos nodos. Caso contrário precisamos nos preocupar em restaurar o balanço da árvore. A restauração deste balanço é efetuada através do que denominamos ROTAÇÕES na árvore. As operações de rotação são melhor entendidas através de exemplos. Inicialmente vamos considerar a seguinte árvore (os números ao lado dos nodos são o FB de cada nodo):



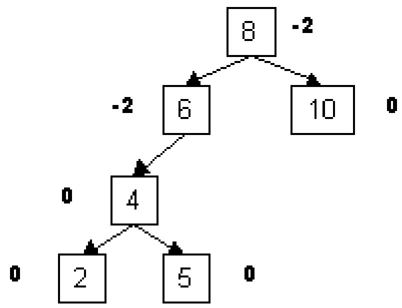
A árvore acima está balanceada, como podemos observar pelos FB de cada nodo. Os casos possíveis de desbalanceamento são 2. Veremos cada um deles.

Tipo 1 - ROTAÇÃO DUPLA

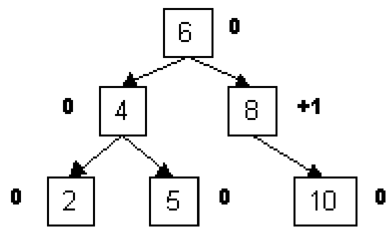
Suponha que agora queremos inserir o número 5 na árvore acima. A inserção produziria a seguinte árvore:



Observe que o nodo 8 tem FB -2 e tem um filho com FB +1. Neste caso o balanceamento é atingido com duas rotações, também denominada ROTAÇÃO DUPLA. Primeiro rotaciona-se o nodo com FB 1 para a esquerda. A árvore ficaria:



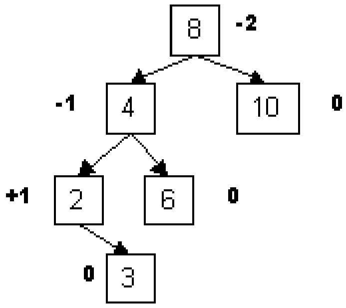
A seguir rotaciona-se o nodo que tinha FB -2 na direção oposta (direita neste caso). A árvore ficaria:



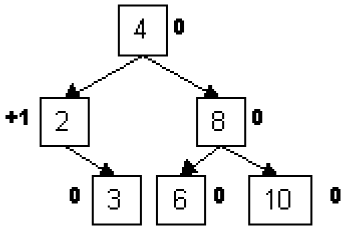
Observe que os FB nos nodos voltaram a ficar dentro do esperado das árvores AVL. O caso simétrico ao explicado acima acontece com os sinais de FB trocados, ou seja, um nodo com FB +2 com um filho com FB -1. Também utilizariamos uma rotação dupla, mas nos sentidos contrários, ou seja, o nodo com FB -1 seria rotacionado para a direita e o nodo com FB +2 seria rotacionado para a esquerda.

Tipo 2 - ROTAÇÃO SIMPLES

Suponha que queremos inserir o nodo 3 na árvore inicial. A inserção produziria a seguinte árvore:



A inserção do nodo 3 produziu um desbalanço no nodo 8 verificado pelo FB -2 neste nodo. Neste caso, como os sinais dos FB são os mesmos (nodo 8 com FB -2 e nodo 4 com FB -1) significa que precisamos fazer apenas uma ROTAÇÃO SIMPLES à direita no nodo com FB -2. No caso simétrico (nodo com FB 2) faríamos uma rotação simples à esquerda. Após a rotação simples a árvore ficaria:



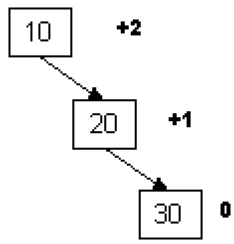
Observe mais uma vez que os FB estão dentro do esperado para mantermos a propriedade de balanceamento de árvores AVL.

CONSTRUINDO UMA ÁRVORE AVL

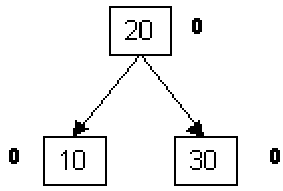
A descrição acima pode ser resumida no seguinte algoritmo em pseudo-código para construção de uma árvore AVL:

1. Insira o novo nodo normalmente (ou seja, da mesma maneira que inserimos numa ABP);
2. Iniciando com o nodo pai do nodo recém-inserido, teste se a propriedade AVL é violada neste nodo, ou seja, teste se o FB deste nodo é maior do que **abs(1)**. Temos aqui 2 possibilidades:
    2.1 A condição AVL foi violada
        2.1.1 Execute as operações de rotação conforme for o caso (Tipo1ou Tipo 2)
        2.1.2 Volte ao passo 1
    2.2 A condição AVL não foi violada. Teste pela condição AVL o pai do nodo testado por último (ou seja, retorne ao passo 2). Se o nodo recém-testado não tem pai, ou seja, é o nodo raiz da árvore, volte para inserir novo nodo (Passo 1)

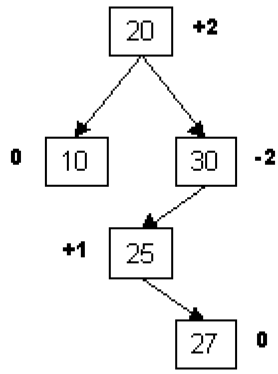
O importante a observar neste algoritmo é que o teste por desbalanço inicia com o último nodo inserido, e não com o nodo raiz! Vamos ver um exemplo de construção de uma árvore AVL com os seguintes números: <10,20,30,25,27>. A inserção dos 3 primeiros números resulta na seguinte árvore:



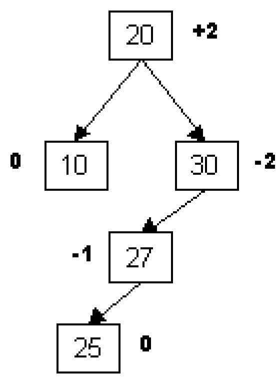
Após a inserção do elemento 30 a árvore fica desbalanceada. O caso acima é do Tipo 2. Fazemos uma rotação para a esquerda no nodo com FB 2. A árvore resultante fica:



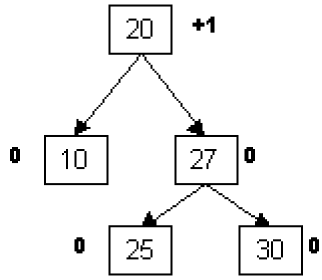
O passo seguinte é inserir os nodos 25 e 27. A árvore fica desbalanceada apenas após a inserção do nodo 27, exemplificado abaixo:



Este caso é do Tipo 1. O nodo 30 tem FB -2 e o seu nodo filho tem FB 1. Precisamos efetuar uma rotação dupla, ou seja, uma rotação simples à esquerda do nodo 25, resultando



seguida de uma rotação simples à direita do nodo 30, resultando:



e a árvore está balanceada.

Árvores-B (B-Trees)

Estrutura proposta por R. Bayer e E. Mc Creight (o B de B-Tree vem de Bayer), em 1970 É bastante utilizada para manipulação de registros em arquivos pelo seu bom desempenho. Uma B-Tree é uma árvore de pesquisa *M-ária* que possui as seguintes características:

- a raiz ou é uma folha ou possui ao menos dois filhos;
- cada nó intermediário, com exceção da raiz, possui entre M e 2M elementos;
- o caminho entre a raiz e qualquer folha possui o mesmo comprimento;
- um nó intermediário com k elementos (sendo k, no máximo, igual a 2M) possui k+1 filhos.

Uma árvore-B pode ser de dois tipos, com relação à distribuição dos elementos na sua estrutura:

1) os elementos estão distribuídos na estrutura da árvore (nós intermediários e folhas), neste caso temos uma **árvore B**;
2) os elementos estão apenas nas folhas, e nos nós intermediários tem-se índices para acessar as folhas, neste caso temos uma **árvore B+**.

Exemplo de árvore B+

Exemplo de uma árvore-B+ de ordem 2 (M=2), com elementos apenas nas folhas (e com índices de acesso aos elementos nos nós intermediários):

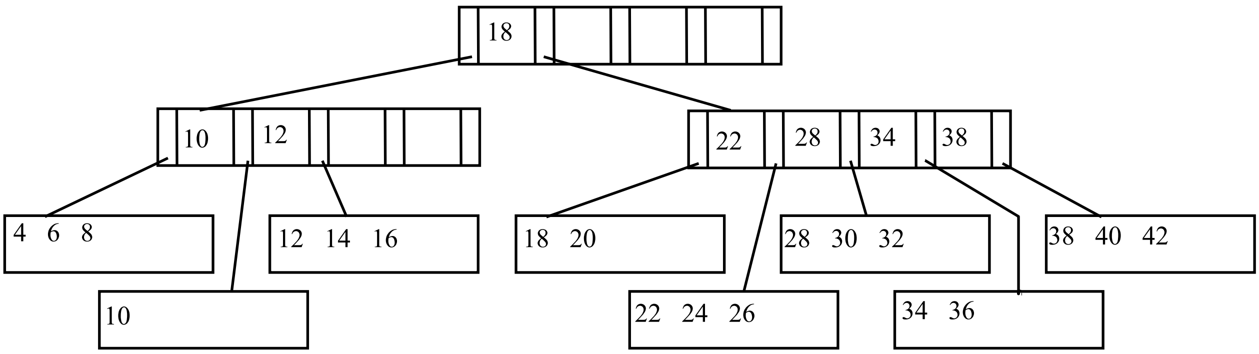


Figura 1 - Exemplo de árvore B+ de ordem 2

Neste exemplo, a capacidade máxima do nó intermediário é de 4 elementos (2M=4): um nó intermediário com 2 elementos (k elementos) possui 3 filhos (k+1), um nó intermediário com 3 elementos (k elementos) possui 4 filhos (k+1), um nó intermediário com 4 elementos (k elementos) possui 5 filhos (k+1).

Em uma árvore balanceada pode-se definir as folhas e os nós intermediários com capacidades diferentes, utilizando a árvore como estrutura intermediária de acesso (as folhas poderiam representar um bloco, ou alguma outra estrutura em memória secundária, por exemplo). A árvore poderia servir apenas para otimizar o acesso, como uma estrutura auxiliar. A árvore B+ é um exemplo típico de estrutura auxiliar, utilizada como índice. No exemplo anterior (figura 1) tem-se um caso destes, e pode-se observar que os elementos estão todos nas folhas, sendo que nos nós intermediários tem-se apenas chaves de acesso às folhas (por isso alguns deles estão repetidos nos nós intermediários). Em cada um dos nós intermediários, o primeiro elemento (do nó) contém o menor valor acessado através do seu segundo filho, o segundo elemento contém o menor valor acessado através do seu terceiro filho, o terceiro elemento contém o menor valor acessado através do seu quarto filho, e assim por diante. Maiores detalhes sobre este tipo de estrutura poderão ser encontradas no item sobre B+ Trees.

Estrutura dos nós da árvores B

Considerando que os nós intermediários armazenarão os elementos (ou apenas as chaves dos elementos, como na figura 1) (**e**) e apontadores para os filhos (**a**), cada nó de uma árvore-B possui a seguinte estrutura:

a_0	e_1	a_1	e_2	a_2	...	a_{n-1}	e_n	a_n

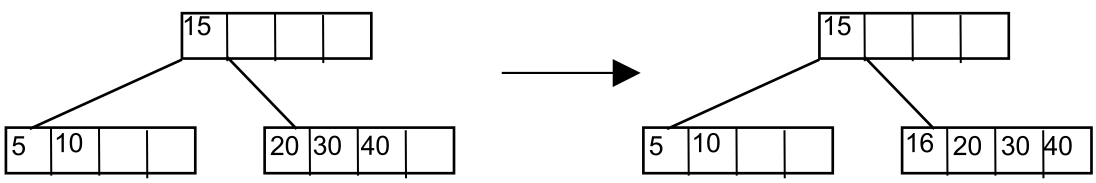
As chaves dos elementos (**e**) estão ordenadas: $\mathbf{e_1<e_2...<e_n}$, e as chaves de cada um dos filhos, apontados por $\mathbf{a_0,a_1,...,a_n}$, também, sendo que os elementos (ou as chaves) existentes no filho apontado por $\mathbf{a_0}$ são menores que os do filho apontado por $\mathbf{a_1}$, que são menores que os do filho apontado por $\mathbf{a_2}$, e assim respectivamente.

- OPERAÇÕES: as operações a seguir (consulta, inserção e remoção) são válidas para as árvores-B com elementos distribuídos na estrutura da árvore. Para as operações em árvores-B com elementos só nas folhas, ver a explicação das árvores 2-3.

a) CONSULTA: para pesquisar um elemento **x** (ou de chave **x**), compara-se **x** com os nós: se $\mathbf{x<e_1}$, deve-se continuar a pesquisa através de $\mathbf{a_0}$; se $\mathbf{e_1 <= x < e_2}$, deve-se continuar a pesquisa através de $\mathbf{a_1}$, e assim sucessivamente.

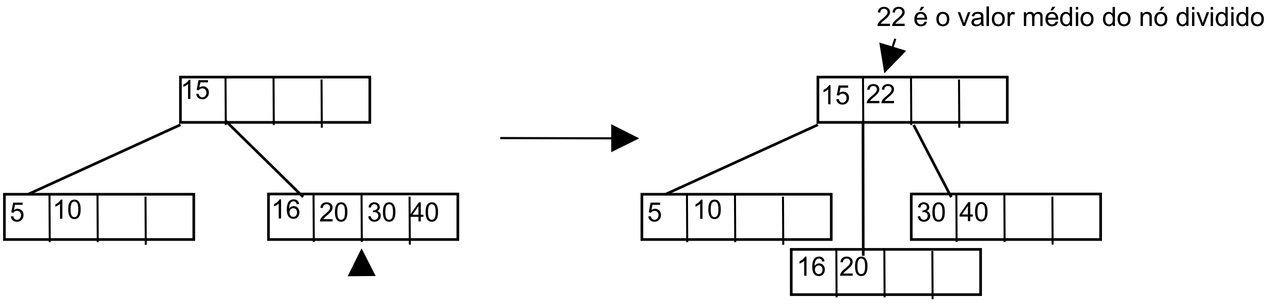
b) INSERÇÃO: após ser verificado que o elemento a ser inserido não pertence à árvore, é preciso inseri-lo em um nó folha, de forma a preservar a ordenação da árvore.

Exemplo: inserir o 16, considerando que os elementos ficam distribuídos em toda a estrutura da árvore (não somente nos nodos folha, mesmo que inicialmente eles sejam inseridos sempre nas folhas).



Se não houver mais espaço para a inserção em um nó (overflow), é necessário dividi-lo em dois (“SPLIT”). O elemento que possuir o valor médio do nó é movido para o seu nó-pai (nó antecessor).

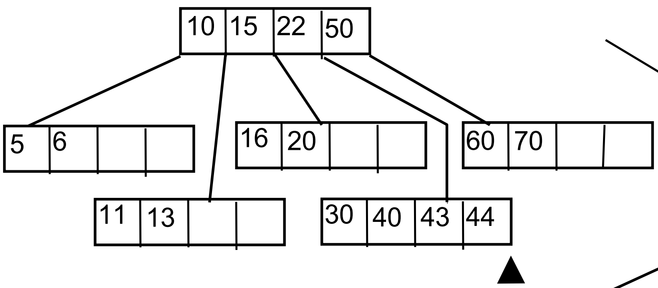
Exemplo: inserir o 22

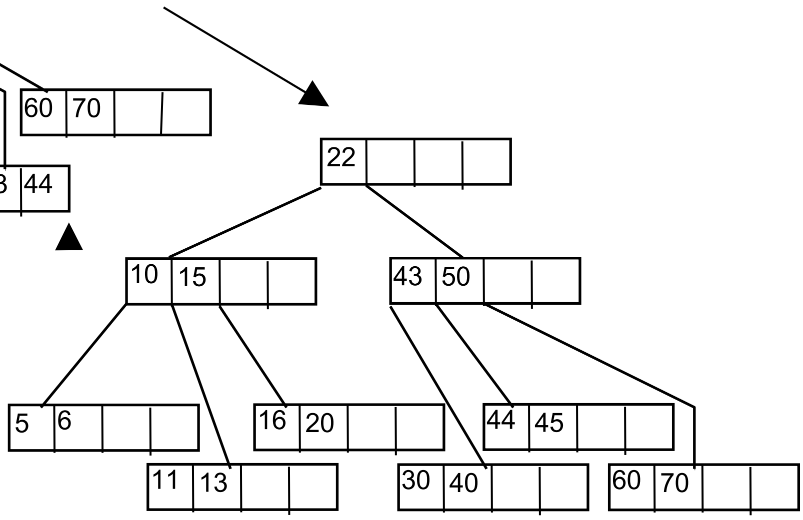


Eventualmente a inserção de um elemento no seu nó antecessor pode causar a divisão dele também, ocasionando uma propagação de divisões. Em um caso extremo, esta propagação poderia atingir a raiz da árvore, causando sua divisão.

Depois do 22, foram inseridos: 6, 11, 13 (*split*, sobe o 10), 60, 70, 50 (**split**, sobe o 50), 43, 44.

Exemplo: inserir o 45





- sobe para o nível anterior o 43, o que causa a divisão da raiz.

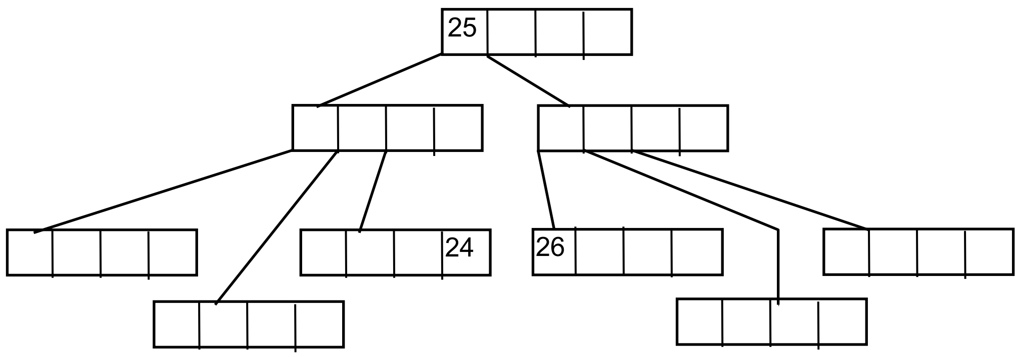
A divisão da raiz determina a criação de uma nova raiz, e em tal situação a árvore “cresce” um nível. ***Uma árvore-B cresce então das folhas em direção à raiz.***

b) REMOÇÃO: a remoção de um elemento pode se enquadrar em um de dois casos:

1. o elemento a ser removido está em um nó folha;
2. o elemento não está em um nó folha. Neste caso ele deve ser substituído por um **elemento adjacente** que esteja em uma folha.

Em qualquer caso, deve ser verificado se não há violação da condição de que todo nó, com exceção da raiz, deve ter entre M e 2M elementos. Em uma remoção, muitas vezes um nó fica com menos que M elementos (“underflow”). Neste caso, é necessário *juntar* (“MERGE”) nós para manter a integridade da B-Tree.

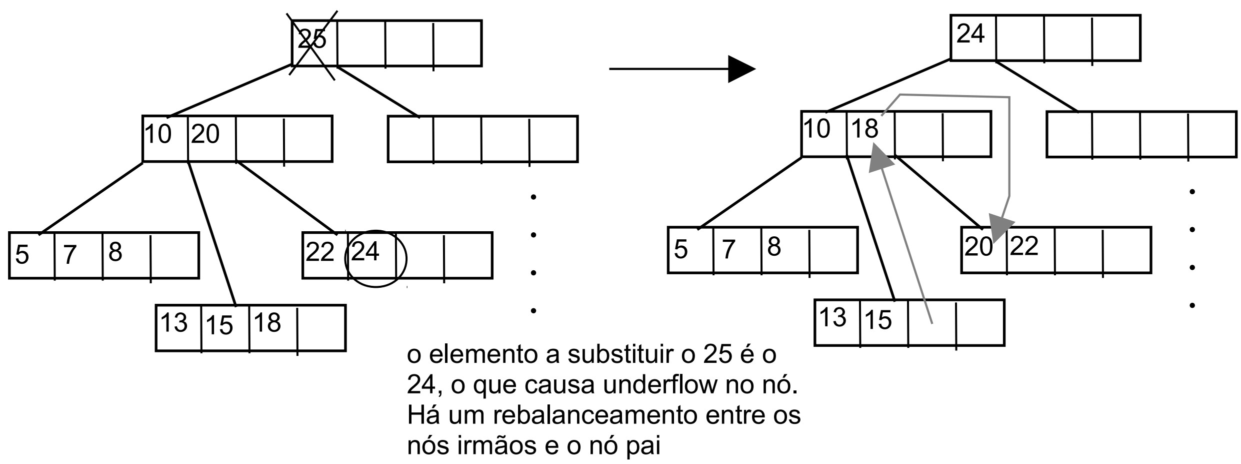
Elemento adjacente: o processo para encontrá-lo é semelhante ao da remoção em árvore binária. Pode-se, a partir do ponteiro à esquerda do elemento a ser removido, descer pelos ponteiros mais à direita em direção a uma folha e fazer a substituição pelo elemento mais à direita (no exemplo, substituir o 25 pelo 24), ou , a partir do ponteiro da direita, descer pelos ponteiros mais à esquerda e fazer a substituição pelo elemento mais à esquerda (no exemplo, substituir o 25 pelo 26).



No caso de “underflow” em um nó:

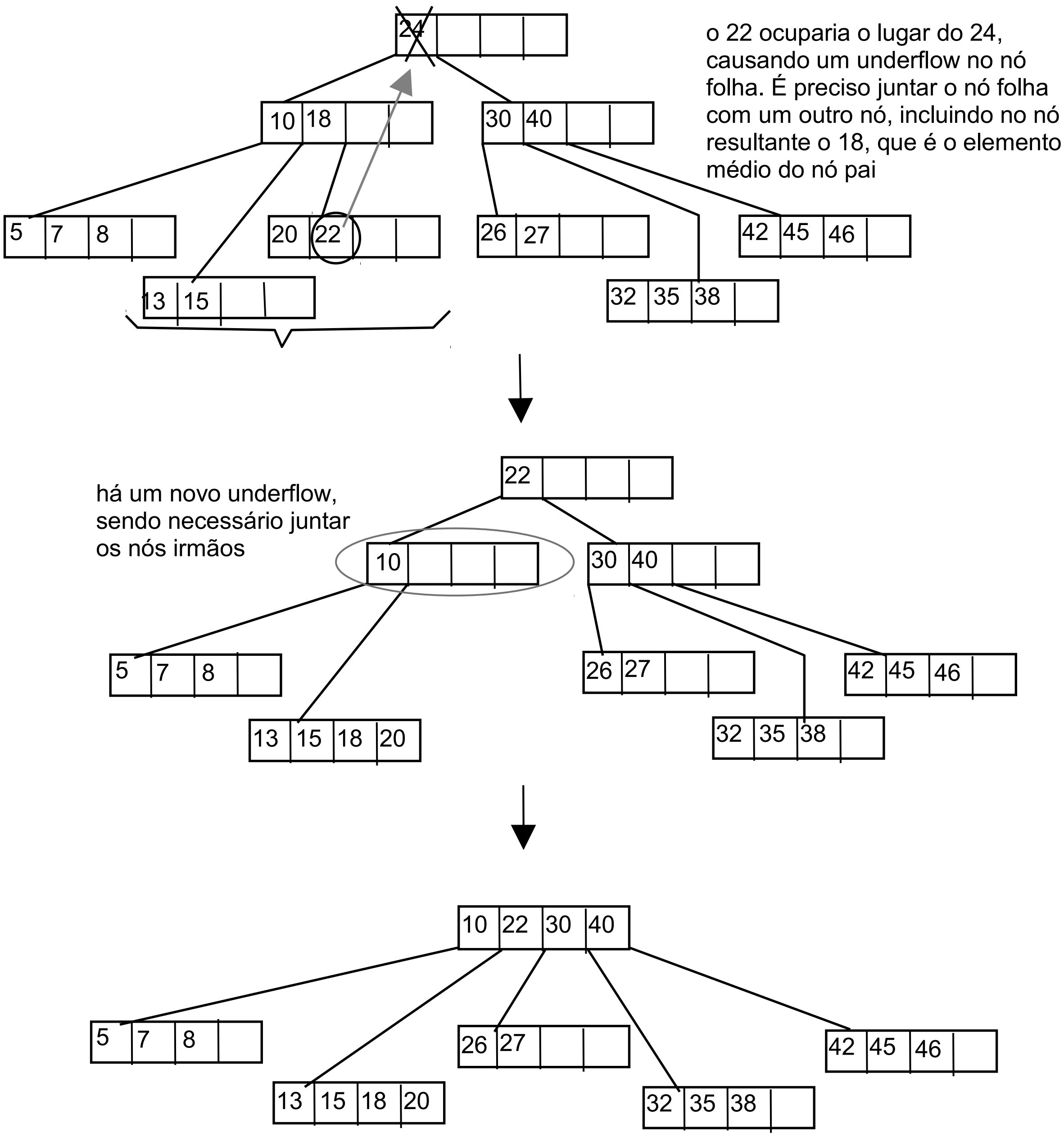
1º caso) existem nós irmãos e é possível anexar um elemento de um deles sem causar novo underflow. Neste caso, é preciso redistribuir os elementos entre os nós-irmãos e o nó-pai para manter a ordenação.

Exemplo: remover o 25



2º caso) existem nós irmãos e a anexação de um elemento de um deles causaria um novo underflow. Neste caso, é preciso *juntar* ("MERGE") os dois nós (o nó com underflow e seu irmão mais próximo), adicionando também o elemento médio que está no nó-pai entre os elementos agora em um mesmo nó.

Exemplo: remover o 24



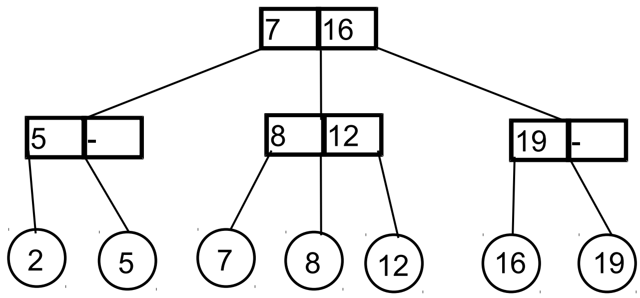
O underflow em um nó pode se propagar em direção à raiz, podendo causar o surgimento de uma nova raiz com a consequente diminuição da altura da árvore em um nível (conforme apresentado no exemplo acima).

Árvores 2-3 (2-3 Trees)

São árvores-B+ de ordem 1, com as seguintes propriedades:

- cada nó tem 2 ou 3 filhos (grau máximo igual a 3; ordem igual a 1);
- o comprimento de um caminho da raiz até qualquer folha é o mesmo;
- os elementos ficam armazenados nas folhas, os nós intermediários possuem apenas "chaves" de acesso (aqui é considerado desta forma, mas há uma outra abordagem de árvores 2-3 que mantém os elementos inteiros nos nós intermediários);
- é mantido o critério de ordenação: se um elemento **a** está à esquerda de um elemento **b**, então **a<b**.

Em cada nó que não é folha, guarda-se a chave do menor elemento descendente do segundo filho deste nó. Se este nó tiver um terceiro filho, guarda-se também a chave do menor elemento descendente do terceiro filho deste nó. No exemplo a seguir, os nós intermediários são identificados por retângulos e as folhas por círculos:



Operações:

a) INSERÇÃO: o elemento deve ser inserido se ele ainda não estiver na árvore. Poderão ocorrer as seguintes situações:

- se o nó intermediário que será seu pai tiver apenas 2 filhos, basta inseri-lo como terceiro filho, na ordem adequada, e atualizar o nó intermediário-pai acresentando o segundo índice.
- se o nó intermediário já tiver 3 filhos (folhas), não é possível inserir um quarto. Neste caso, é necessário dividir (*split*) o nó intermediário em dois. Um deles ficará com dois elementos (dois dos três filhos já existentes) e o outro ficará com os outros dois (o terceiro dos três filhos já existentes mais o elemento a ser inserido).

Os nós intermediários antecessores terão que refletir essas mudanças nos índices que guardam. Isso poderá implicar também a divisão dos nós intermediários, chegando inclusive até a uma divisão da raiz da árvore.

b) REMOÇÃO: ao remover uma folha, é possível deixar seu nó-pai com apenas um filho (o que não pode acontecer, uma vez que cada nó pode ter no mínimo 2 filhos). Neste caso, é necessário reestruturar a árvore:

- se este nó-pai que ficou com apenas um filho for a raiz, ela deve ser removida e este filho se tornará a raiz;
- se não for a raiz:

a) caso o nó-pai tenha um irmão (também nó intermediário) com três filhos, passa-se um destes para o nó-pai com apenas um filho, e cada um deles ficará com dois filhos;

b) caso o nó-pai tenha um irmão com apenas dois filhos, transfere-se o filho único para este irmão, e remove-se o nó-pai. Se este irmão agora tornar-se filho único, repete-se a seqüência de procedimentos acima, recursivamente, para o pai do nó-pai (removido).

c) CONSULTA: compara-se **x**, a chave do elemento a ser consultado, com **y**, a 1ª chave guardada por um nó intermediário. Se **x<y**, segue-se a pesquisa pelo seu primeiro filho; se **x>y** ou **x=y** e este nó intermediário só tiver dois filhos, segue-se a pesquisa pelo seu segundo filho; se **x>y** ou **x=y** e houver um terceiro filho, compara-se x com **z**, a 2ª chave guardada pelo nó intermediário. Neste caso, se **x<z** segue-se a pesquisa pelo segundo filho e se **x>z** ou **x=z**, segue-se a pesquisa pelo terceiro filho.

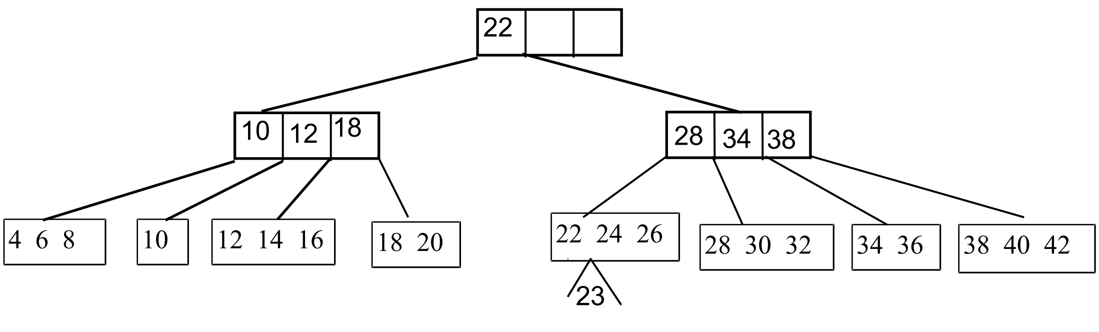
Árvores-B* (B*-Trees)

É uma árvore-B, cujos nós têm que possuir no mínimo 2/3 de sua ocupação total (ao invés de metade de sua ocupação total, na árvore-B). Isto significa que, ao invés de dividir um nó completamente preenchido em dois nós preenchidos pela metade, é preciso possuir dois nós completamente preenchidos para então dividi-los em três novos nós, cada um preenchido em 2/3 de seu espaço.

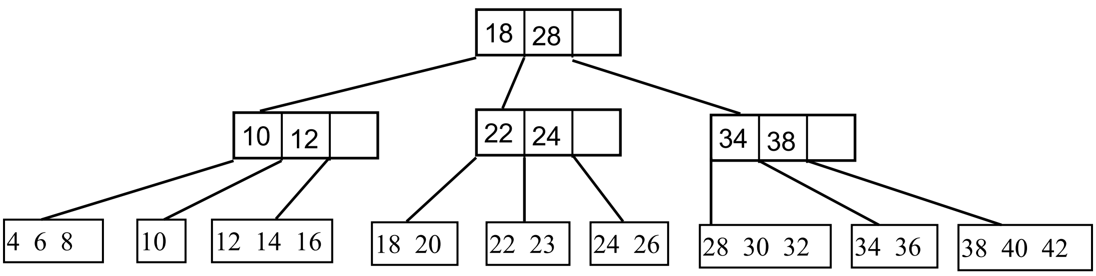
As operações de CONSULTA, INSERÇÃO e REMOÇÃO são semelhantes às realizadas em B-Trees, com a diferença que o “split”, na inserção, e o “merge”, na remoção, seguem respectivamente a restrição de que é preciso de dois nós cheios para serem divididos em três e que no mínimo tem que preencher 2/3 da ocupação de cada nó.

b) INSERÇÃO: é similar à inserção na árvore 2-3: a 1ª chave de um nó intermediário será a chave do menor elemento de seu segundo filho; a 2ª chave será a chave do menor elemento de seu segundo filho, e assim por diante.

Semelhante à árvore 2-3, se não houver mais espaço para o elemento a ser inserido (num conjunto de dois nós) , os nós têm que ser divididos em três novos nós, e a atualização das chaves refletida no nó-pai. Se no nó-pai também não houver mais espaço o processo se repete recursivamente. Caso seja necessário dividir a raiz, cria-se uma nova raiz com dois filhos, e esta é a única situação em que um nó pode ter um número menor de filhos.



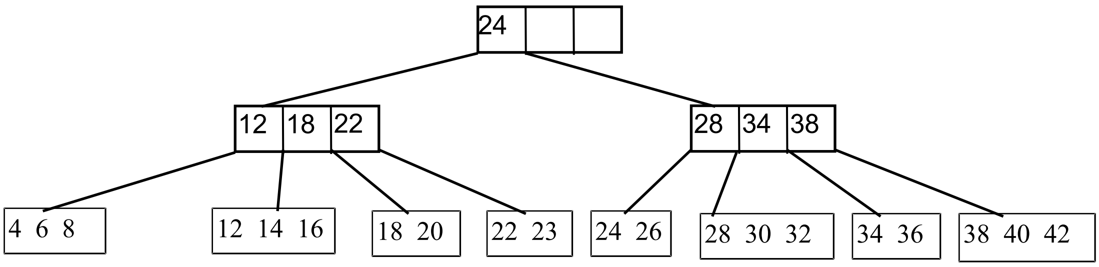
Árvore-B* após a inserção de 23:



c) REMOÇÃO: ao remover um nó (folha), é necessário atualizar as chaves e eventualmente os apontadores de seus ancestrais. Além disso, é preciso garantir que nenhum nó possua menos que 2/3 de sua ocupação. Se isso ocorrer, é preciso ajustar a árvore para corrigir a situação, da seguinte forma:

- se este nó possuir um irmão com mais do que 2/3 de sua ocupação, passa-se um destes itens do irmão para o nó que sofreu a remoção; caso o irmão esteja no limite de itens (exatamente 2/3 de sua ocupação), faz-se a fusão do nó que sofreu a remoção com o irmão. Em ambos os casos, tem-se que fazer a atualização do pai, que também pode vir a ficar com menos de 2/3 de sua ocupação, e o processo tem que se repetir recursivamente.

Árvore-B após a remoção de 10:



Neste exemplo, o menor número de filhos de um nó é igual a 3, com exceção da raiz que pode ter no mínimo 2.

Bibliografia:

[1] Aho, A.V.; Hopcroft,J.E.;Ullman,J.D. ***Data Structures and Algorithms***. Addison-Wesley Publishing Company, 1983.

[2] Furtado,A.L;Santos,C.S. ***Organização de Banco de Dados***. Editora Campus, 1987

[3] Veloso,P.... ***Estruturas de Dados***. Editora Campus, 1984

[4] Terada,R. ***Desenvolvimento de Algoritmos e Estruturas de Dados.*** Editora McGraw-Hill, 1991.

[5] Szwarcfiter,J.L.;Markenzon, L. ***Estruturas de Dados e seus Algoritmos.*** Editora LTC, 1994.

[6] Wirth, N. ***Algorithms + Data Structures = Programs.*** Prentice-Hall, Inc., 1976.