8.3.1. Algoritmos para Manipulação da Árvore AVL

Estrutura de um nodo na árvore AVL:

```
Estrutura nodoAVL {
   info  : tipo_info;
   esq   : *nodoAVL;
   dir   : *nodoAVL;
   alt   : inteiro;
};
```

O campo **alt** deve ser atualizado recursivamente quando um nodo for inserido ou deletado.

Para o retorno da altura de uma subárvore necessitamos de uma função especial, já que um nodo pode não ter um ou ambos os filhos (lembre-se que a pesquisa para garantir a condição-AVL é realizada perguntando-se a altura das duas subárvores):

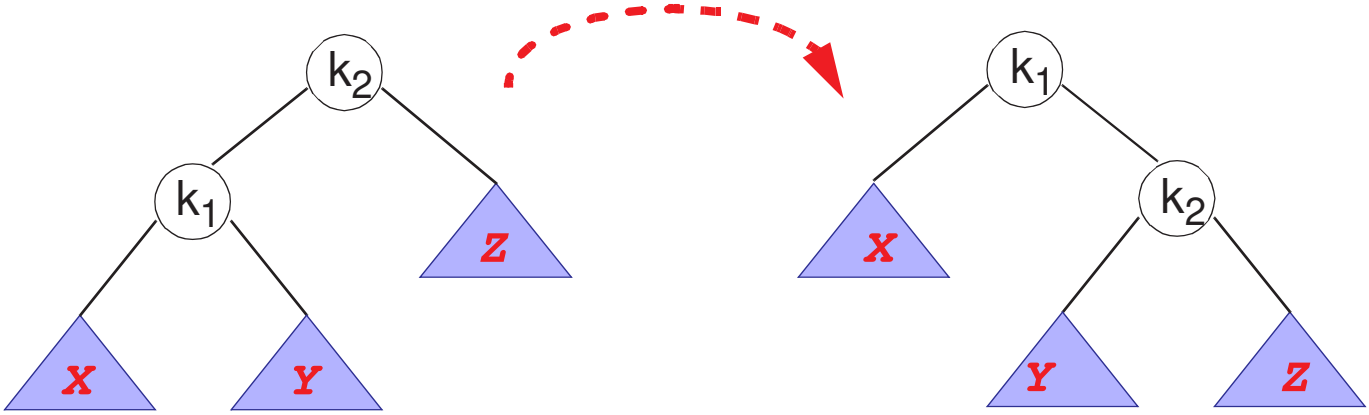
```
inteiro altura( subárvore : *nodoAVL )
   início
      se (subárvore for NULO)
         retorne -1; /* A altura de uma subárvore
                        inexistente é definida como -1 */
      senão
         retorne subárvore->alt;
      fim se
   fim
```

Além deste, teremos também algoritmos para rotação das subárvores:

8.3.1.1. Algoritmos para efetuar a Rotação Simples

Rotação simples à esquerda:
Esta função será chamada somente se k_2 possuir um filho à esquerda.

- Executa uma rotação entre um nodo k_2 e seu filho da esquerda.
- Atualiza as alturas.
- Retorna a nova raiz.

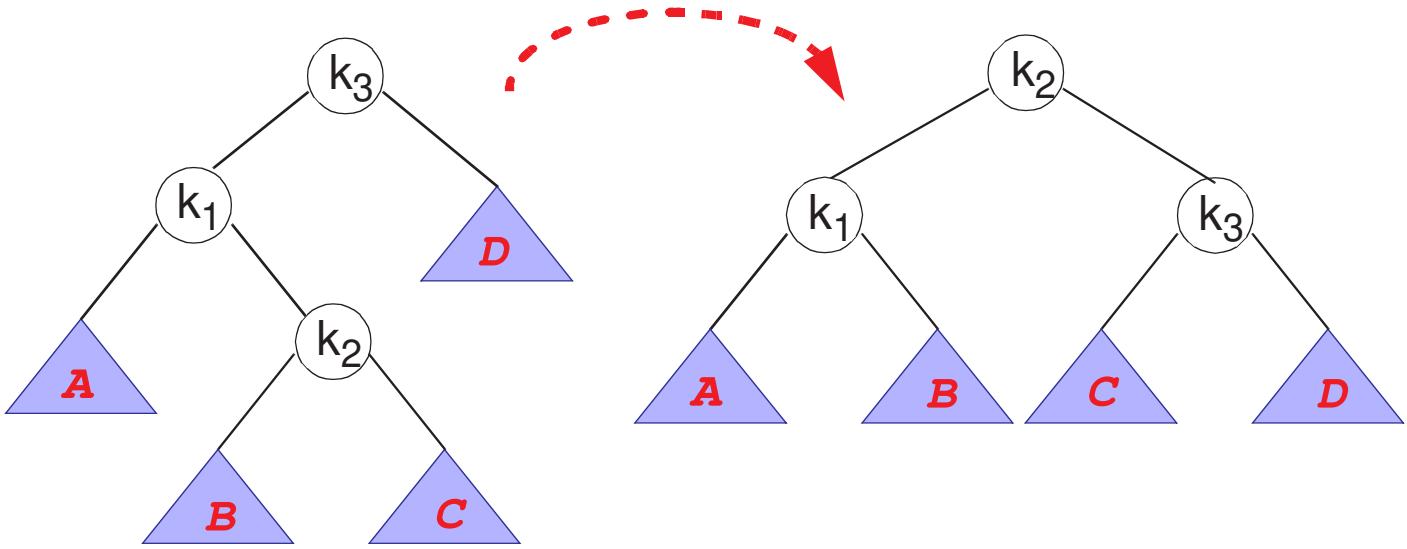



```
nodoAVL *simp_roda_esq(k2 *nodoAVL)
   variáveis locais
      k1 : *nodoAVL;
   início
      k1 <- k2->esq;
      k2->esq <- k1->dir;
      k1->dir <- k2;

      /* atualize alturas */
      k2->alt <- max(altura(k2->esq), altura(k2->dir));
      k1->alt <- max(altura(k1->esq), k2->alt);

      retorne k1; /* nova raiz da subárvore */
   fim
```

A rotação à direita é simétrica e pode ser facilmente implementada.

8.3.1.2. Dupla Rotação à Esquerda

Esta função será chamada somente se k3 possuir um filho à esquerda e se o filho à esquerda de k3 possuir um filho à direita. Ela realiza a rotação e atualiza as alturas das subárvores rotacionadas.




```
nodoAVL *dup_roda_esq(k3 : *nodoAVL)
   início
      /* Rotacione entre k1 e k2 */
      k3->esq <- simp_roda_dir( k3-> esq );

      /* Rotaciona entre k3 e k2 */
      retorne ( simp_roda_esq( k3 );
   fim
```

Da mesma forma que com a rotação simples, a rotação dupla à direita é a operação simétrica da rotação à esquerda e é facilemente implementada.

8.3.1.3. Inserção em Árvore AVL

```
nodoAVL *inserçãoAVL(info : tipo_info, arv : *nodoAVL, pai : *nodoAVL)
variáveis
    arv_rodada : *nodoAVL;
início
    se (arv é NULO) então /* Folha: aloca novo nodo */
       arv <- aloca novo nodo;
       se (arv é NULO) então retorne ERRO;
       arv->info <- info;   arv->alt <- 0;
       arv->esq <- NULO;    arv->dir <- NULO;
    senão
       se (info < arv->info) então
          arv->esq <- inserçãoAVL(info, arv->esq, arv);
          se ((altura(arv->esq) - altura(arv->dir) > 1)
             se (info < arv->esq->info) então
                arv_rodada <- simp_roda_esq(arv);
             senão
                arv_rodada <- dup_roda_esq(arv);
             se (pai->esq = arv) então
                pai->esq <- arv_rodada;
             senão
                pai->dir <- arv_rodada;
             fim se
          senão
             arv->alt <- max( altura(arv->esq), altura(arv->dir));
          fim se
       senão
          se (info > arv->info) então
             /* caso simétrico para árvore direita */
          senão
             retorne ERRO: „chave já está na árvore“
          fim se
       fim se
    fim se
    retorne arv;
fim
```

8.3.1.4. Exclusão

A exclusão pode ser implementada utilizando-se o algoritmo de exclusão para a árvore de busca binária dado anteriormente, com apenas algumas modificações:

- Depois de excluir um nodo, tenho de balancear o nodo que agora ocupa o seu lugar.
- Tenho de retornar recursivamente atualizando a altura das subárvores.

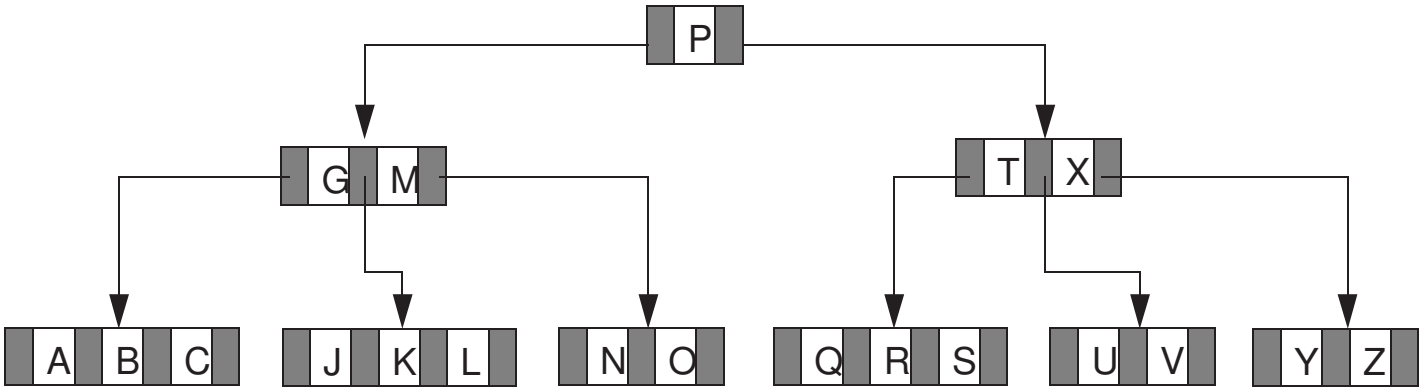
8.3.1.5. Comentários finais:

A primeira chamada da inserção pode ser feita passando-se um ponteiro nulo como pai da árvore ou então através de uma rotina que tem por objetivo somente chamar a inserçãoAVL da forma adequada:

```
nodoAVL *insere( info : tipo_info, arv : *nodoAVL )
   início
      retorne inserçãoAVL( info, arv, NULO );
   fim
```

8.4. Árvore B

- São árvores de pesquisa balanceadas especialmente projetadas para a pesquisa de informação em discos magnéticos e outros meios de armazenamento secundário.
- Minimizam o número de operações de movimentação de dados (escrita/leitura) numa pesquisa ou alteração.
- O grau de um nó pode ser alto.
- Podem ser consideradas como uma generalização natural das árvores de pesquisa binárias.
- Exemplo:



8.4.1. Definição

Uma árvore-B de grau mínimo t >= 2 é uma árvore com as seguintes propriedades:

1. Cada nó possui os seguintes campos:
   <NChaves,Folha,P_1,<K_1,P_{R1}>,P_2,<K2,P_{R2}>,...,<K_{q-1},P_{Rq-1}>,P_q>
   onde:
   K: chaves
   Folha: flag indicando se o nó é folha ou não
   q < 2t
   P_i: ponteiro para o nó filho **i**
   P_{Ri}: ponteiro para o elemento de dados na memória ou para um registro em disco onde se encontra a chave **K_i**

2. Em cada nó as chaves estão em ordem crescente.

3. Cada chave **K_i** separa as subárvores **P_i** e **P_{i+1}**. As chaves da subárvores Pi são menores do que **K_i** e as chaves da subárvore **P_{i+1}** são maiores do que **K_i**.

4. Existem limites inferior e superior ao número de chaves em cada nó:
   (a) Cada nó, exceto a raiz, contém no mínimo **t-1** chaves (**t** filhos)
   (b) Cada nó pode ter no máximo **2t-1** chaves (**2t** filhos).

5. Todos os nós-folha estão no mesmo nível.

8.4.2. Inserção em uma Árvore-B

O algoritmo de inserção em uma Árvore-B é bastante simples:

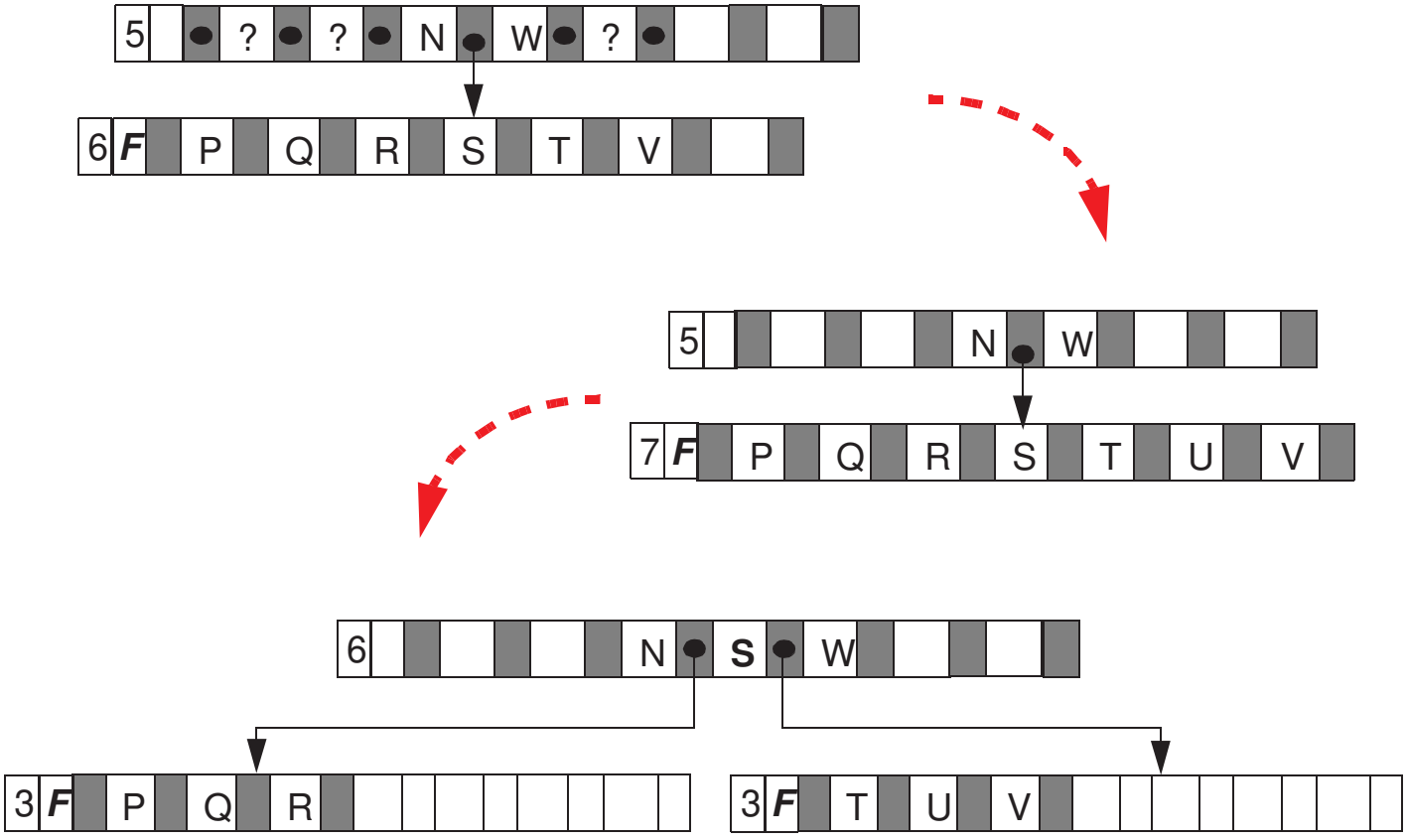
0. A inserção é sempre feita nas folhas.

1. Desça a árvore em busca da chave.

2. Verifique se o nó está cheio: se contém **`2t-1`** chaves.

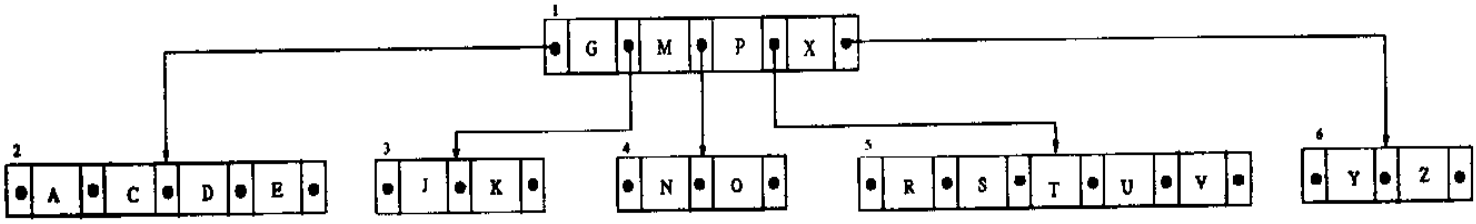
3. Se o nó estiver cheio:
   3.a. Dividir o nó ao meio.
   3.b. Colocar a chave do meio no nó pai.
   3.c. Se este também encher, repetir o processo recursivamente.

Exemplo simples: Inserção de „U“



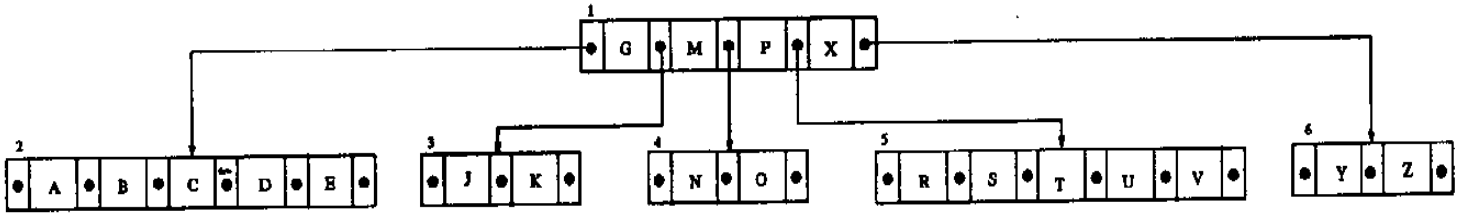
8.4.2.1. Seqüência de inserções (exemplo):

- Considere a árvore abaixo com **t=3**:



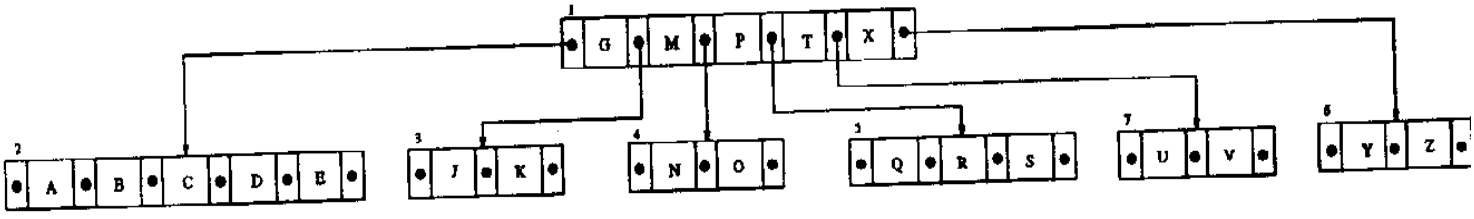
Árvore original

- Inserir B:



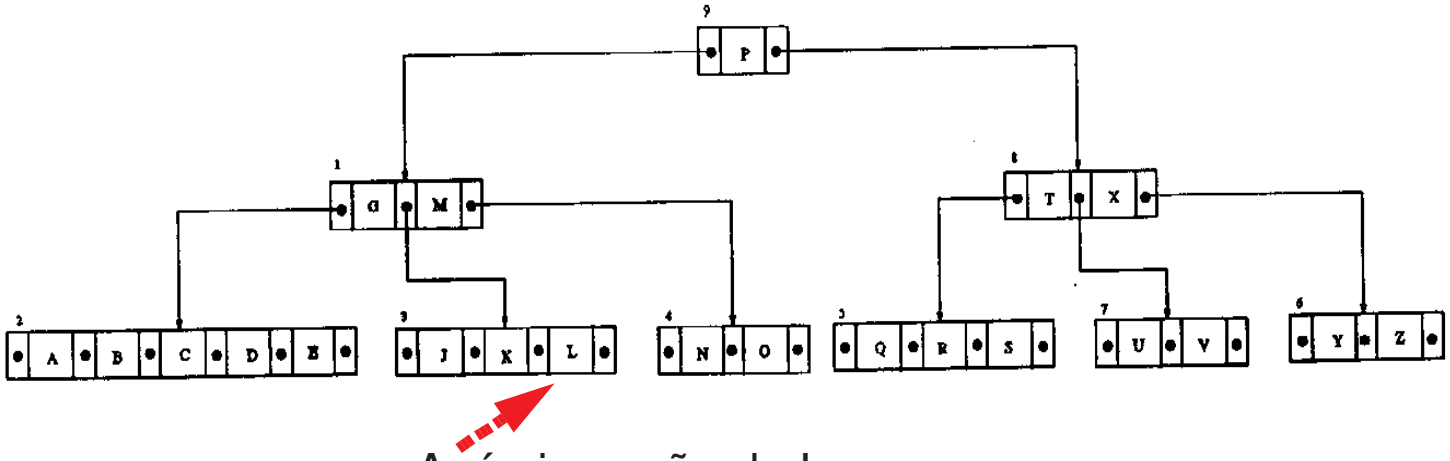
Após inserção de B

- Inserir Q:
  - o nó 5 é dividido em dois (5 e 7)
  - a chave T é movida para o nó pai
  - Q é inserido no nó de endereço 5



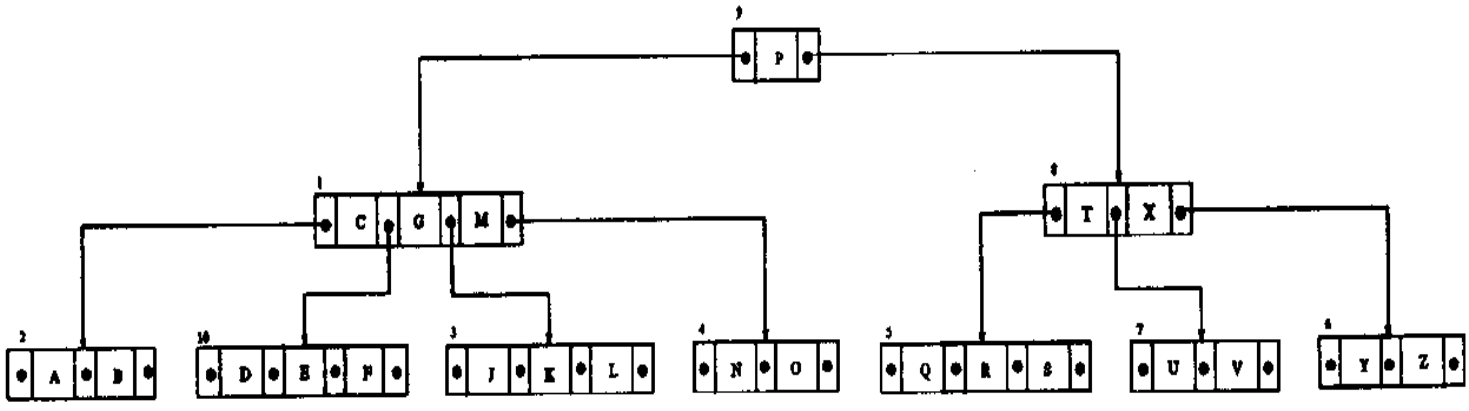
Após inserção de Q

- **Inserir L**:
  - a raiz está cheia
  - a raiz é dividida em dois nós (1 e 8)
  - a árvore cresce em altura com o deslocamento de P para o novo nó 9
  - Q é inserido no nó de endereço 3



Após inserção de L

- **Inserir F**:
  - o nó 2 está cheio e é dividido em dois nós (2 e 10)
  - a chave C é movida para o nó pai
  - F é inserido no nó de endereço 10.



Após inserção de F

8.4.3. Remoção em Árvore-B

A remoção é a operação mais complexa em uma árvore-B:

Passos iniciais:

1. Desça a árvore em busca da chave.

2. Verifique se haverá underflow (**`t-1`**). Se ocorrer, lembre-se que será necessário fundir o nó com algum irmão, seguindo regras bem definidas.

Regras de Remoção (remover a chave K da subárbvore X):

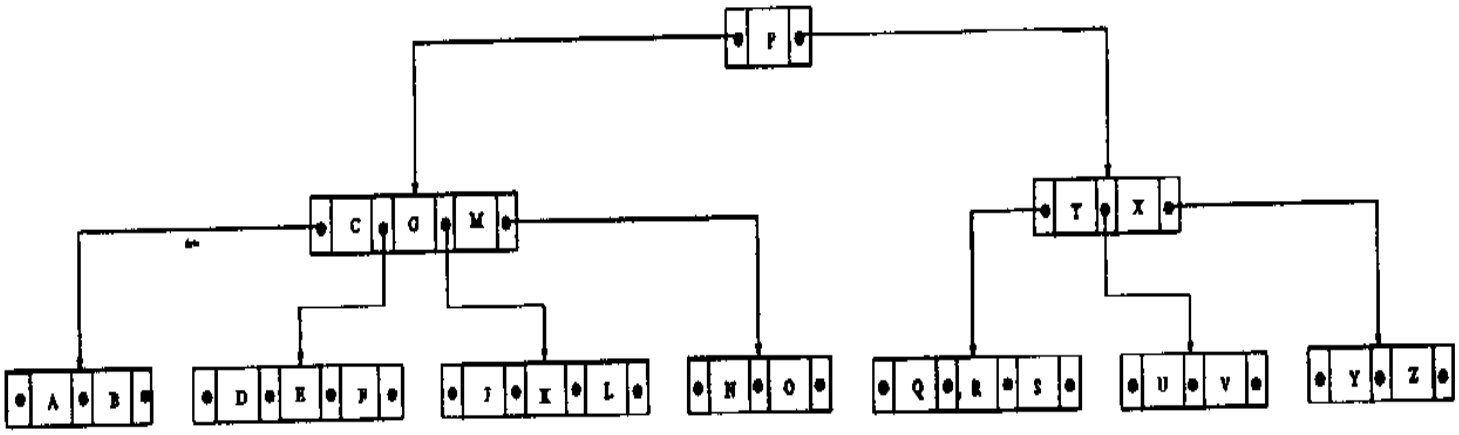
1. Se K está em X e X é folha: **`Remova`**

2. Se K está em X e X é um nodo interno:
   (a) Se a subárvore Y predecessora de X possui pelo menos **`t`** chaves:
   - determine a chave predecessora K' em Y
   - substitua K por K'
   - recursivamente exclua K' da subárvore Y
   (b) Se a subárvore Z sucessora de X possui pelo menos **`t`** chaves:
   - determine a chave sucessora K' em Z
   - substitua K por K'
   - recursivamente exclua K' da subárvore Z
   (c) Se as subárvores Y e Z possuem apenas **`t-1`** chaves:
   - funda K e todo o nodo Z em Y
   - retire K e o apontador para Z de X
   - libere o nó Z
   - recursivamente exclua K da subárvore Y.

3. Se K não está em X e X é um nó interno:
   - determine a raiz **`P`$_i$`[X]`** da subárvore que deve conter K
   - **SE** **`P`$_i$`[X]`** tem apenas **`t-1`** chaves:
   (a) Se um dos vizinhos de **`P`$_i$`[X]`** tem pelo menos t chaves:
      - mova uma chave de X descendo para **`P`$_i$`[X]`**
      - mova uma chave do vizinho de **`P`$_i$`[X]`** subindo para X
      - mova o apontador apropriado do vizinho para **`P`$_i$`[X]`**
      - recursivamente exclua K de **`P`$_i$`[X]`**
   (b) Se **`P`$_i$`[X]`** e seus vizinhos têm apenas t-1 chaves:
      - funda **`P`$_i$`[X]`** com um dos seus vizinhos e com a chave K_i de X
      - remova o parK_i e P_i de X
      - libere o nó apontado por P_{ri}
      - recursivamente exclua K do nodo em que foi feita a junção
   - **SENÃO** :
      Exclua recursivamente K de **`P`$_i$`[X]`**

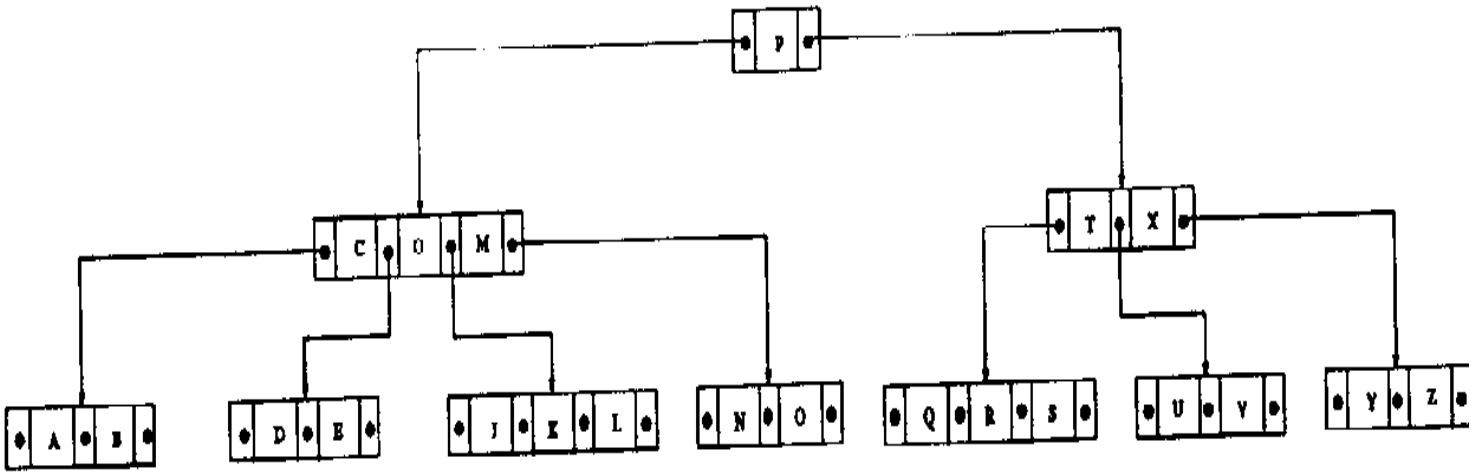
8.4.3.1. Exemplo de Exclusão

Considere a árvore abaixo:



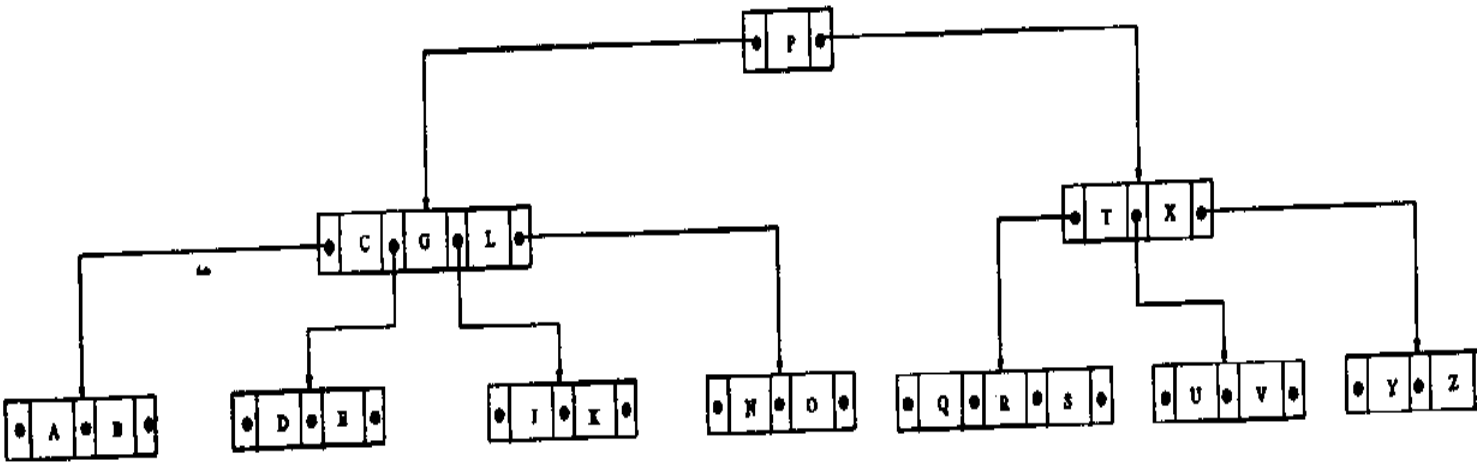
Árvore Inicial

- Caso 1: Remoção do F



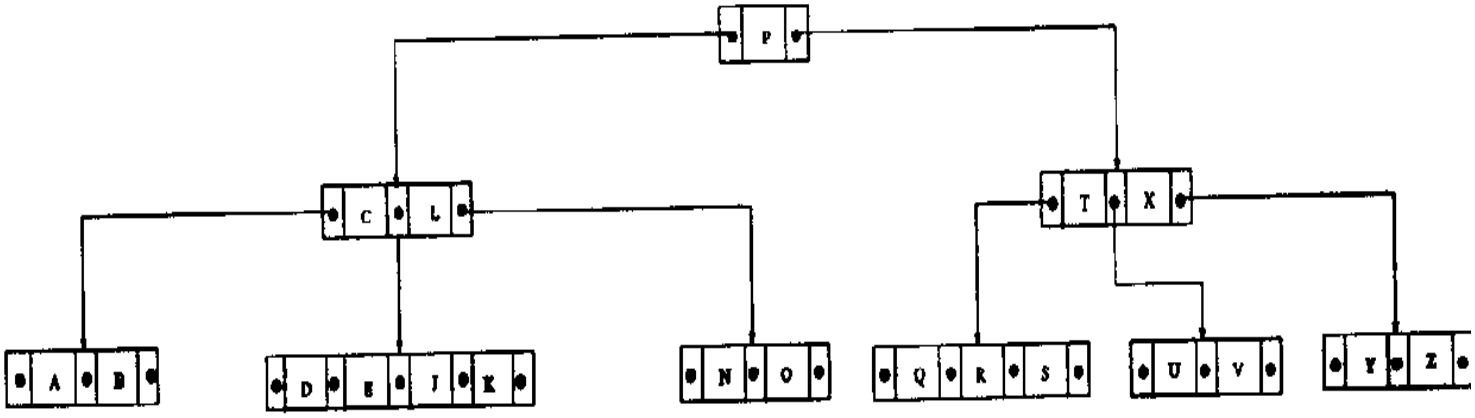
após remoção do F

- Caso 2a: Remoção do M



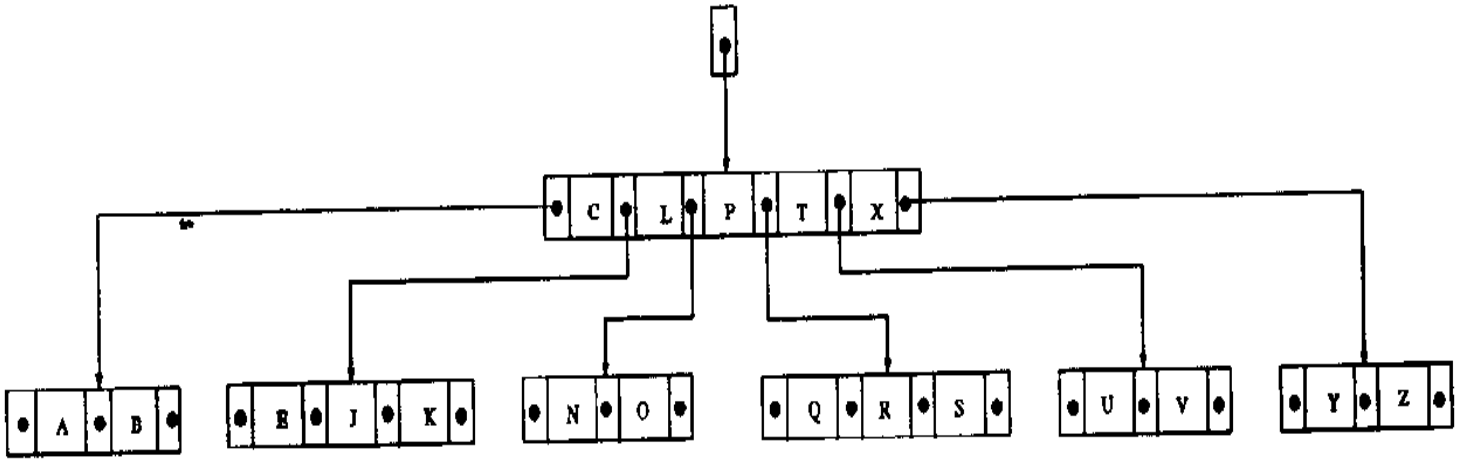
após remover o M

- Caso 2c: Remover o G



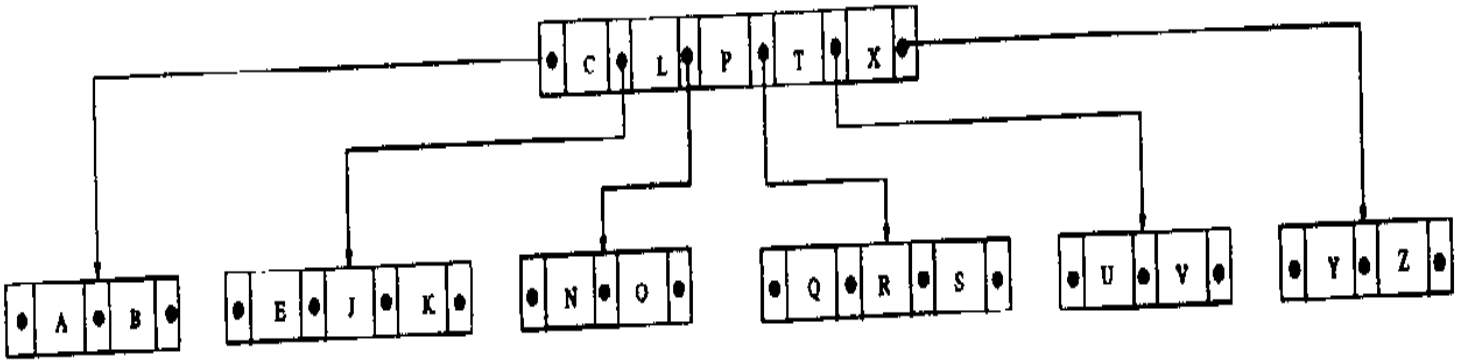
após remover o G

- Caso 3b: Remover D



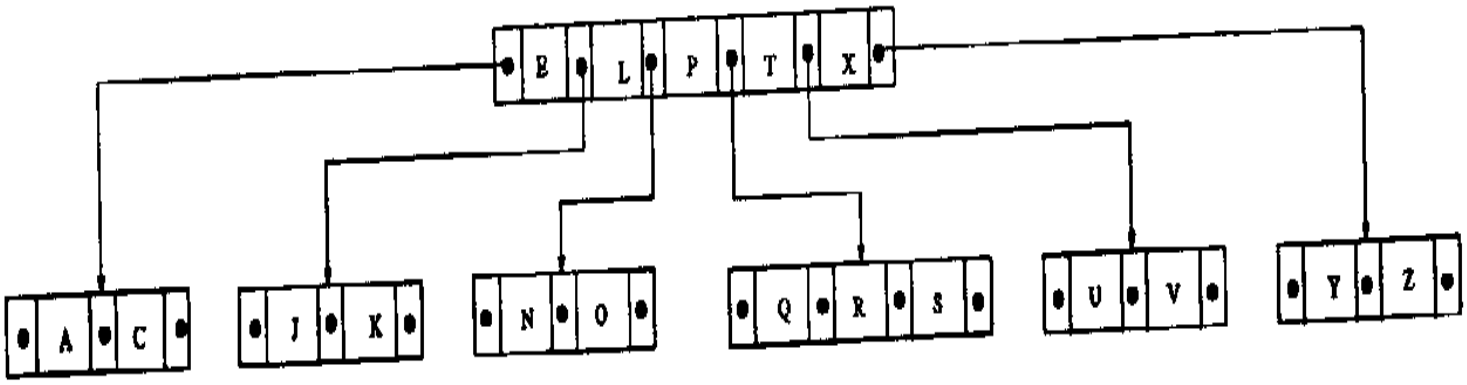
após remover D

- Caso 3B: Árvore diminui de altura



após reduzir-se a altura

- Caso 3a: Remover B



-

8.5. Exercícios

1. Mostre sob a forma de desenho, tanto para uma árvore AVL como para uma árvore-B com t=3, os resultados de:
a) criar uma árvore vazia e inserir os seguintes elementos nesta ordem: F,S,Q,K,C,L,H,T,V,P,A,B,X,D.
b) excluir desta árvore os elementos K,C e L.

2. Descubra uma estratégia para imprimir o conteúdo das árvores acima em ordem alfabética. Teste os algoritmos preordem, emordem e pós-ordem nos seus desenhos e veja se você pode usar um deles. Quais as modificações necessárias para percorrer uma árvore B?

3. **Trabalho:** Implemente uma das duas árvores dadas (AVL ou B) em C++. O programa deverá ser capaz de ler um arquivo ASCII com os seguintes dados de pessoas, nesta forma: <nome>|<telefone>|<endereço>. O caracter separador de campos será o caracter „|“. O arquivo conterá uma pessoa por linha.
Serão fornecidos 3 arquivos:
- 1 com uma lista (grande) de pessoas a serem incluídas
- 1 com uma lista de pessoas a serem excluídas
- 1 com uma lista de pessoas a serem incluídas depois da exclusão.

A equipes que implementarem árvore B deverão combinar entre si o grau da árvore, para que cada equipe implemente uma árvore com grau diferente. O programa deverá, durante a inclusão e a exclusão escrever em um arquivo os seguintes dados, em ascii, neste formato: <número de elementos na árvore> <tempo necessário para efetuar uma operação de inclusão ou de exclusão>. O tempo necessário para ler um dado do arquivo não deverá ser considerado.
Ao final o programa deverá chamar o programa unix **`gnuplot`** com o arquivo de tempos como parâmetro para que um gráfico dos tempos seja mostrado na tela.
O programa deverá também listar todos os nomes em ordem na tela.