Programa 5.7 *Procedimento para inserir na árvore*

```
procedure Insere (x: TipoRegistro; var p: TipoApontador);
begin
  if p = nil
  then begin
       new (p);
       p^.Reg := x;  p^.Esq := nil;  p^.Dir := nil;
       end
  else if x.Chave < p^.Reg.Chave
       then Insere (x, p^.Esq)
       else if x.Chave > p^.Reg.Chave
            then Insere (x, p^.Dir)
            else writeln ('Erro: Registro ja existe na arvore')
end; { Insere }
```

A árvore de pesquisa mostrada na Figura 5.2 pode ser obtida quando as chaves são lidas pelo Programa 5.8, na ordem 5, 3, 2, 7, 6, 4, 1, 0, sendo 0 a marca de fim de arquivo.

Programa 5.8 *Programa para criar a árvore*

```
program CriaArvore;
type TipoChave = integer;
{-- Entra aqui a definicao dos tipos do Programa 5.4 --}
var Dicionario: TipoDicionario;
    x         : TipoRegistro;
{-- Entram aqui os Programas 5.6 e 5.7--}
begin
Inicializa (Dicionario);
read (x.Chave);
while x.Chave > 0 do
  begin
  Insere (x, Dicionario);
  read (x.Chave);
  end;
end.
```

A última operação a ser estudada é **Retira**. Se o nó que contém o registro a ser retirado possui no máximo um descendente, então a operação é simples. No caso de o nó conter dois descendentes, o registro a ser retirado deve ser primeiro substituído pelo registro mais à direita na subárvore esquerda, ou pelo registro mais à esquerda na subárvore direita. Assim, para retirar o registro com chave 5 na árvore da Figura 5.2, basta trocá-lo pelo registro com chave 4 ou pelo registro com chave 6, e então retirar o nó que recebeu o registro com chave 5.

O Programa 5.9 mostra a implementação da operação Retira. O procedimento recursivo Antecessor somente é ativado quando o nó que contém o registro a ser retirado possui dois descendentes. Essa solução elegante é utilizada por Wirth (1976, p. 211).

Programa 5.9 *Procedimento para retirar x da árvore*

```
procedure Retira (x: TipoRegistro; var p: TipoApontador);
var Aux: TipoApontador;

  procedure Antecessor (q: TipoApontador; var r: TipoApontador);
  begin
    if r^.Dir <> nil
    then Antecessor (q, r^.Dir)
    else begin
         q^.Reg := r^.Reg;
         q := r;  r := r^.Esq;
         dispose (q)
         end;
  end; { Antecessor }

begin {--  Retira --}
  if p = nil
  then writeln ('Erro: Registro nao esta na arvore')
  else if x.Chave < p^.Reg.Chave
       then Retira (x, p^.Esq)
       else if x.Chave > p^.Reg.Chave
            then Retira (x, p^.Dir)
            else if p^.Dir = nil
                 then begin Aux := p;  p := p^.Esq;  dispose(Aux); end
                 else if p^.Esq = nil
                      then begin Aux:=p;  p:=p^.Dir;  dispose(Aux); end
                      else Antecessor (p, p^.Esq);
end; { Retira }
```

Após construída a árvore, pode ser necessário percorrer todos os registros que compõem a tabela ou arquivo. Existe mais de uma ordem de **caminhamento** em árvores, mas a mais útil é a chamada ordem de **caminhamento central**. Assim como a estrutura da árvore, o caminhamento central é mais bem expresso em termos recursivos, a saber:

1. caminha na subárvore esquerda na ordem central;
2. visita a raiz;
3. caminha na subárvore direita na ordem central.

Uma característica importante do caminhamento central é que os nós são visitados em ordem lexicográfica das chaves. Percorrer a árvore da Figura 5.2