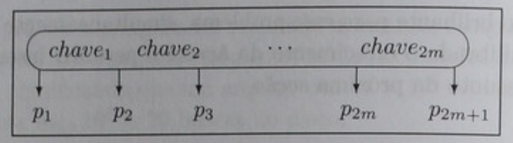

Figura 6.9 *Nó de uma árvore B de ordem m com $2m$ registros.*

A estrutura de dados árvore B será utilizada para implementar o tipo abstrato de dados Dicionario, com as operações Inicializa, Pesquisa, Insere e Retira. A estrutura e a **representação** do TipoDicionario são apresentadas no Programa 6.3, no qual mm significa $2m$. O procedimento Inicializa é extremamente simples, conforme ilustra o Programa 6.4.

Programa 6.3 *Estrutura do dicionário para árvore B*

```
type TipoRegistro   = record
                        Chave: TipoChave;
                        {outros componentes}
                      end;
     TipoApontador  = ^TipoPagina;
     TipoPagina     = record
                        n: 0..mm;
                        r: array [1..mm] of TipoRegistro;
                        p: array [0..mm] of TipoApontador
                      end;
     TipoDicionario = TipoApontador;
```

Programa 6.4 *Procedimento para inicializar uma árvore B*

```
procedure Inicializa (var Dicionario: TipoDicionario);
begin
  Dicionario := nil;
end; { Inicializa }
```

Um procedimento Pesquisa para árvore B é semelhante ao algoritmo Pesquisa para árvore binária de pesquisa, conforme pode ser visto no Programa 6.5. Para encontrar um registro x, primeiro compare a chave que rotula o registro com as chaves que estão na página raiz até encontrar o intervalo no qual ela se encaixa. Depois, siga o apontador para a subárvore correspondente ao intervalo citado e repita o processo recursivamente, até que a chave procurada seja encontrada ou então uma página folha seja atingida (no caso, um apontador nulo). Na implementação do Programa 6.5, a localização do intervalo em que a chave se encaixa

é obtida por meio de uma pesquisa sequencial. Entretanto, essa etapa pode ser realizada de forma mais eficiente, utilizando-se pesquisa binária (vide Seção 5.2).

Programa 6.5 *Procedimento para pesquisar na árvore B*

```
procedure Pesquisa (var x: TipoRegistro; Ap: TipoApontador);
var i: Integer;
begin
  if Ap = nil
  then writeln ('Registro nao esta presente na arvore')
  else with Ap^ do
       begin
       i := 1;
       while (i < n) and (x.Chave > r[i].Chave) do i := i + 1;
       if x.Chave = r[i].Chave
       then x := r[i]
       else if x.Chave < r[i].Chave
            then Pesquisa (x, p[i-1])
            else Pesquisa (x, p[i])
       end;
end; { Pesquisa }
```

Vamos ver agora como **inserir** novos registros em uma árvore B. Em primeiro lugar, é preciso localizar a página apropriada em que o novo registro deve ser inserido. Se o registro a ser inserido encontra seu lugar em uma página com menos de $2m$ registros, o processo de inserção fica limitado àquela página. Entretanto, quando um registro precisa ser inserido em uma página já cheia (com $2m$ registros), o processo de inserção pode provocar a criação de uma nova página. A Figura 6.10(b) ilustra o que acontece quando o registro contendo a chave 14 é inserido na árvore (a). O processo é composto pelas seguintes etapas:

1. O registro contendo a chave 14 não é encontrado na árvore, e a página 3 (onde o registro contendo a chave 14 deve ser inserido) está cheia.
2. A página 3 é dividida em duas páginas, o que significa que uma nova página 4 é criada.
3. Os $2m + 1$ registros (no caso são cinco registros) são distribuídos igualmente entre as páginas 3 e 4, e o registro do meio (no caso o registro contendo a chave 20) é movido para a página pai no nível.

No esquema de inserção apresentado anteriormente, a página pai tem de acomodar um novo registro. Se a página pai também estiver cheia, então o mesmo processo de divisão tem de ser aplicado de novo. No pior caso, o processo de divisão pode propagar-se até a raiz da árvore e, nesse caso, ela aumenta sua altura em um nível. É interessante observar que uma árvore B somente aumenta sua altura com a divisão da raiz.