

Considerando o arranjo g da Figura 5.17 vamos ver como empacotar os seis valores de g em apenas 2 bytes. Na chamada do procedimento `AtribuiValor2Bits`, consideremos a atribuição de `Valor = 2` na posição `Indice = 4` de g (no caso, $g[4] = 2$). No primeiro comando do procedimento o *byte* que vai receber `Valor = 2 = (10)2` é determinado por $i = \text{Indice} \div 4 = 4 \div 4 = 1$ (no caso o segundo *byte*). A posição dentro do *byte* é determinada pelo comando seguinte, onde $\text{Pos} = \text{Indice} \bmod 4 = 4 \bmod 4 = 0$ (isto é, nos dois *bits* menos significativos do *byte*). A seguir, $\text{Pos} = \text{Pos} * 2$ porque cada valor ocupa 2 *bits* do *byte*. A seguir, `not (3 shl Pos) = not ((00000011)2 shl 0) = (11111100)2`. Logo, $g[i] \text{ and } (11111100)_2$ zera os 2 *bits* a atribuir. Finalmente, o comando $g[i] \text{ or } (\text{Valor} \text{ shl } \text{Pos})$ realiza a atribuição e o *byte* fica como $(\text{XXXXXX}10)_2$, onde X representa 0 ou 1.

Programa 5.39 Rotula grafo e atribui valores para o arranjo g usando 2 bits por entrada

```
{— Assume que todas as entradas de 2 bits do vetor —}
{— g foram iniciadas com o valor 3 —}

procedure AtribuiValor2Bits (var g : Tipog;
                           Indice: integer;
                           Valor : byte);
var i, Pos : integer;
begin
  i := Indice div 4;
  Pos := (Indice mod 4);
  Pos := Pos * 2; { Cada valor ocupa 2 bits }
  g[i] := g[i] and (not(3 shl Pos)); { zera os dois bits a atribuir }
  g[i] := g[i] or (Valor shl Pos); { realiza a atribuicao }
end; {AtribuiValor2Bits}

function ObtemValor2Bits (var g : Tipog;
                          Indice: Integer) : byte;
var i, Pos: Integer;
begin
  i := Indice div 4;
  Pos := (Indice mod 4);
  Pos := Pos * 2; { Cada valor ocupa 2 bits }
  ObtemValor2Bits := (g[i] shr Pos) and 3;
end; {ObtemValor2Bits}

{— Atribuiig —}
Procedure Atribuiig (var Grafo: TipoGrafo;
                    var L : TipoArranjoArestas;
                    var g : Tipog);
var i, j, u : integer;
    v : TipoValorVertice;
    a : TipoAresta;
    Soma : integer;
    valorg2bits: integer;
    Visitado : array[0..MAXNUMVERTICES] of boolean;
```

Continuação do Programa 5.39

```
begin
  if (grafo.r <= 3) { valores de 2 bits requerem r <= 3}
  then begin
    for i := Grafo.NumVertices - 1 downto 0 do
      begin
        AtribuiValor2Bits (g, i, grafo.r);
        Visitado[i] := false;
      end;

    for i := Grafo.NumArestas - 1 downto 0 do
      begin
        a := L[i];
        Soma := 0;
        for v := Grafo.r - 1 downto 0 do
          if not Visitado[a.Vertices[v]]
          then begin
            Visitado[a.Vertices[v]] := true;
            u := a.Vertices[v];
            j := v;
          end
          else Soma := Soma + ObtemValor2Bits (g, a.Vertices[v]);
        valorg2bits := (j - Soma) mod grafo.r;
        AtribuiValor2Bits (g, u, valorg2bits);
      end;
    end;
  end; { Atribuiig }
```

Para obter a função *hash* perfeita hp , agora usando 2 *bits* por entrada de g , basta substituir no Programa 5.38 o comando

```
v := v + g[a[i]];
```

pelo comando

```
v := v + ObtemValor2Bits(g, a[i]);
```

Para obter uma função *hash* perfeita mínima nós precisamos de uma estrutura de dados que permita reduzir o intervalo da tabela de $[0, M - 1]$ para $[0, N - 1]$, conforme ilustra a Figura 5.18. Vamos utilizar uma **estrutura de dados sucinta**⁶ que permite computar em tempo constante a função *rank*: $[0, M - 1] \rightarrow [0, N - 1]$ que conta o número de posições atribuídas antes de uma dada posição v em g . Por exemplo, $\text{rank}(4) = 2$ já que as posições 0 e 1 foram atribuídas, uma vez que os valores de $g[0]$ e $g[1]$ são diferentes de 3.

⁶Uma estrutura de dados sucinta usa um espaço muito próximo do limite inferior para a classe do problema em questão, acompanhada de um algoritmo eficiente para a operação de pesquisa.