

No exemplo da Figura 5.17(b), a primeira aresta considerada em  $\mathcal{L}$  é  $a = \{h_0(\text{"mar"}), h_1(\text{"mar"}), h_2(\text{"mar"})\} = \{0, 2, 5\}$ . A Tabela 5.4 mostra os valores das variáveis envolvidas no comando:

for  $v := \text{Grafo.r} - 1$  downto 0 do

Tabela 5.4 Valor das variáveis na execução do Programa 5.37

i	a	v	Visitado	u	j	Soma
2	$\{0, 2, 5\}$	2	False $\rightarrow$ True	5	2	0
		1	False $\rightarrow$ True	2	1	0
		0	False $\rightarrow$ True	0	0	0
1	$\{1, 3, 5\}$	2	True	-	-	3
		1	False $\rightarrow$ True	3	1	3
		0	False $\rightarrow$ True	1	0	3
0	$\{1, 2, 4\}$	2	False $\rightarrow$ True	4	2	0
		1	True	4	2	0
		0	True	4	2	3

O comando após o anel:

$$g[u] := (j - \text{Soma}) \bmod \text{Grafo.r};$$

faz  $g[0] = (0 - 0) \bmod 3 = 0$ . Igualmente, para a aresta seguinte de  $\mathcal{L}$  que é  $a = \{h_0(\text{"jan"}), h_1(\text{"jan"}), h_2(\text{"jan"})\} = \{1, 3, 5\}$ , o comando após o anel faz  $g[1] = (0 - 3) \bmod 3 = -3$ . O comando a seguir:

$$\text{while } g[u] < 0 \text{ do } g[u] := g[u] + \text{Grafo.r};$$

irá fazer  $g[1] = g[1] + 3 = -3 + 3 = 0$ . Finalmente, para a última aresta em  $\mathcal{L}$  que é  $a = \{h_0(\text{"fev"}), h_1(\text{"fev"}), h_2(\text{"fev"})\} = \{1, 2, 4\}$ , o comando após o anel faz  $g[4] = (2 - 3) \bmod 3 = -1$ . O anel que segue faz  $g[4] = g[4] + 3 = -1 + 3 = 2$ .

A partir do arranjo  $g$  podemos obter uma função *hash* perfeita para uma tabela com intervalo  $[0, M - 1]$ . Para uma chave  $k \in S$  a função  $hp$  tem a seguinte forma:

$$hp(k) = h_i(k), \text{ onde } i(k) = (g[h_0(k)] + g[h_1(k)] + \dots + g[h_{r-1}(k)]) \bmod r \quad (5.1)$$

Considerando  $r = 3$ , o vértice escolhido para uma chave  $k$  é obtido por uma das três funções, isto é,  $h_0(k)$ ,  $h_1(k)$  ou  $h_2(k)$ . Logo, a decisão sobre qual função  $h_i(k)$  deve ser usada para uma chave  $k$  é obtida pelo cálculo  $i(k) = (g[h_0(k)] + g[h_1(k)] + g[h_2(k)]) \bmod 3$ . No exemplo da Figura 5.17(b), a chave “jan” está na posição 1 da tabela porque  $(g[1] + g[3] + g[5]) \bmod 3 = 0$  e  $h_0(\text{"jan"}) = 1$ . De forma similar, a chave “fev” está na posição 4 da tabela porque  $(g[1] + g[2] + g[4]) \bmod 3 = 2$  e  $h_2(\text{"fev"}) = 4$ , e assim por diante.

O Programa 5.38 mostra o procedimento para obter a função *hash* perfeita  $hp$ . O procedimento recebe a chave, o valor de  $r$ , os pesos para a função  $h$  do Programa 3.18 e o arranjo  $g$ . O procedimento segue a Eq.(5.1) para descobrir qual foi o vértice da aresta escolhido para a chave.

Programa 5.38 Função de transformação perfeita

```
function hp (Chave : TipoChave;
            r : Tipor;
            var Pesos: TipoTodosPesos;
            var g : Tipog): TipoIndice;
var i, v: integer; a: TipoArranjoVertices;
begin
  v := 0;
  for i := 0 to r - 1 do
    begin
      a[i] := h (Chave, Pesos[i]);
      v := v + g[a[i]];
    end;
  v := v mod r;
  hp := a[v];
end; { hp }
```

Na estrutura de dados mostrada no Programa 5.32 da Seção 5.5.4 o tipo do arranjo  $g$  é *integer*. No Programa 5.38 o tipo do arranjo  $g$  muda para *byte*, e o comando

Tipog = array[0..MAXNUMVERTICES] of integer;

muda para

Tipog = array[0..MAXNUMVERTICES] of byte;

Como sabemos, um *byte* pode armazenar  $2^8 = 256$  valores distintos. Como somente um dos quatro valores 0, 1, 2, ou 3 é armazenado em cada entrada de  $g$ , apenas 2 *bits* são necessários para armazenar quatro valores distintos. O Programa 5.39 mostra como empacotar quatro valores de  $g$  em um *byte*, reduzindo o espaço ocupado para dois *bits* por entrada. Para isso foram criados dois procedimentos: o procedimento *AtribuiValor2Bits* atribui o  $i$ -ésimo valor de  $g$  em uma das quatro posições do *byte* apropriado e a função *ObtemValor2Bits* retorna o  $i$ -ésimo valor de  $g$ . Agora o tipo do arranjo  $g$  permanece como *byte*, mas o comando

Tipog = array[0..MAXNUMVERTICES] of byte;

muda para

$$\text{const MAXGIZE} = \text{Trunc}((\text{MAXNUMVERTICES} + 3)/4)$$

Tipog = array[0..MAXGIZE] of byte;

onde MAXGIZE indica que o arranjo Tipog ocupa um quarto do espaço e o *byte* passa a armazenar 4 valores.

A operação “shl” no procedimento *AtribuiValor2Bits* move os *bits* para a esquerda e entra com zeros à direita (por exemplo,  $b_0, b_1, b_2, b_3, b_4, b_5, b_6, b_7$  shl 6 =  $b_6, b_7, 0, 0, 0, 0, 0, 0$ ). Da mesma maneira, “shr” na função *ObtemValor2Bits* move os *bits* para a direita e entra com zeros à esquerda.