5.5.5 Hashing *Perfeito Usando Espaço Quase Ótimo*

Esta seção apresenta um algoritmo proposto por Botelho (2008) para obter uma função *hash* perfeita que utiliza um número constante de *bits* por chave para descrever a função. Além disso, o algoritmo gera a função em tempo linear e a avaliação da função é realizada em tempo constante. Este é o primeiro algoritmo prático descrito na literatura que utiliza $O(1)$ *bits* por chave para uma função *hash* perfeita mínima sem ordem preservada. Os métodos conhecidos anteriormente ou são empíricos e sem garantia de que funcionam bem para qualquer conjunto de chaves, ou são teóricos e sem possibilidade de se obter uma implementação prática.

Assim como o algoritmo apresentado na Seção 5.5.4 para funções com ordem preservada, o algoritmo utiliza **hipergrafos ou r−grafos** randômicos. Diferentemente do algoritmo anterior, os hipergrafos considerados são r-partidos. Isso permite que r partes do vetor g sejam acessadas em paralelo. Como mostrado em Botelho (2008), as funções mais rápidas e mais compactas são obtidas para hipergrafos com $r = 3$. A Figura 5.17 mostra os passos do algoritmo para $r = 3$, tendo como entrada um conjunto $S = \{\text{jan}, \text{fev}, \text{mar}\}$. A estrutura de dados orientada a arestas apresentada na Seção 7.10.2 é usada, onde cada aresta do hipergrafo é representada por um arranjo de r vértices e para cada vértice existe uma lista de arestas que são incidentes ao vértice.

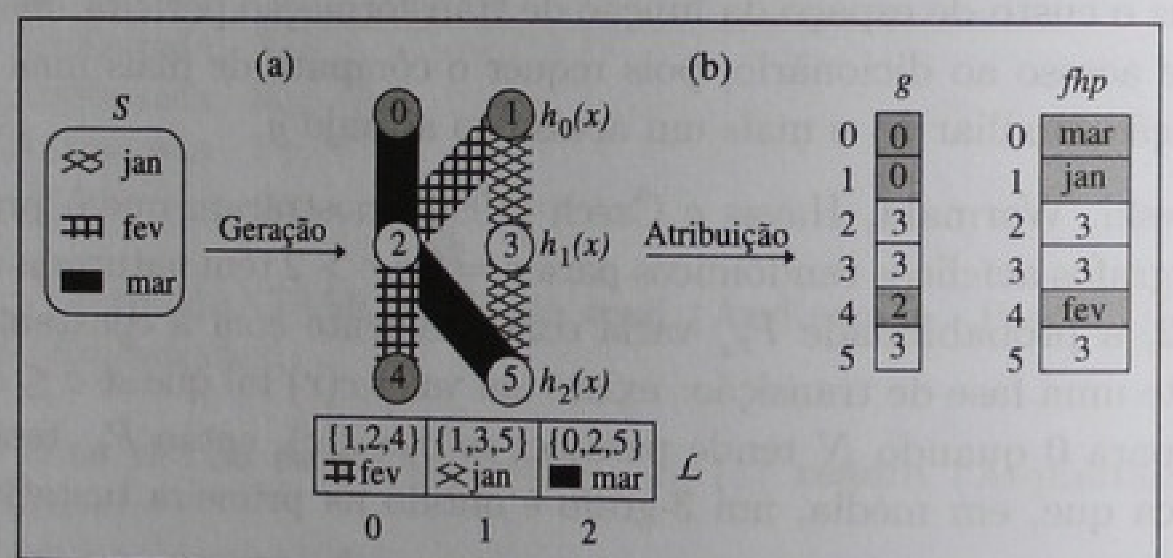

***Figura 5.17** (a) Gera um 3-grafo 3-partido acíclico com $M = 6$ vértices e $N = 3$ arestas e um arranjo de arestas $\mathcal{L}$ obtido no momento de verificar se o hipergrafo é acíclico. (b) Constrói uma função hash perfeita que transforma o conjunto S de chaves para o intervalo $[0, 5]$, sendo representada pelo arranjo $g : [0, 5] \to [0, 3]$ de forma a atribuir univocamente uma aresta a um vértice.*

O passo de geração do hipergrafo na Figura 5.17(a) executa duas tarefas:

1. Utiliza três funções h_0, h_1 and h_2, com intervalos $\{0, 1\}$, $\{2, 3\}$ e $\{4, 5\}$, respectivamente, cujos intervalos não se sobrepõem e por isso o grafo é 3-partido. Essas funções constroem um mapeamento do conjunto S de chaves para o conjunto A de arestas de um r-grafo acíclico $G_r = (V, A)$, onde $r = 3$, $|V| = M = 6$ e $|E| = N = 3$. No exemplo da Figura 5.17(a), "jan" é rótulo da aresta $\{h_0(\text{"jan"}), h_1(\text{"jan"}), h_2(\text{"jan"})\} = \{1, 3, 5\}$, "fev" é rótulo da aresta $\{h_0(\text{"fev"}), h_1(\text{"fev"}), h_2(\text{"fev"})\} = \{1, 2, 4\}$, e "mar" é rótulo da aresta $\{h_0(\text{"mar"}), h_1(\text{"mar"}), h_2(\text{"mar"})\} = \{0, 2, 5\}$.

2. Testa se o hipergrafo randômico resultante contém ciclos por meio da retirada iterativa de arestas de grau 1, conforme mostrado no Programa 7.10. As arestas retiradas são armazenadas em um arranjo $\mathcal{L}$ na ordem em que foram retiradas, o qual é usado no passo seguinte de atribuição. A primeira aresta retirada na Figura 5.17(a) foi $\{1, 2, 4\}$, a segunda foi $\{1, 3, 5\}$ e a terceira foi $\{0, 2, 5\}$. Se terminar com um grafo vazio, então o grafo é acíclico, senão um novo conjunto de funções h_0, h_1 and h_2 é escolhido e uma nova tentativa é realizada.

O passo de atribuição na Figura 5.17(b) produz uma função *hash* perfeita que transforma o conjunto S de chaves para o intervalo $[0, M-1]$, sendo representada pelo arranjo g que armazena valores no intervalo $[0, 3]$. O arranjo g permite selecionar um de três vértices de uma dada aresta, o qual é associado a uma chave k.

O Programa 5.37 mostra o procedimento para obter o arranjo g considerando um hipergrafo $G_r = (V, A)$. As estruturas de dados são as mesmas do Programa 5.32. Para valores $0 \leq i \leq M-1$, o passo começa com $g[i] = r$ para marcar cada vértice como não atribuído e com $Visitado[i] = false$ para marcar cada vértice como não visitado. Seja j, $0 \leq j < r$, o índice de cada vértice u de uma aresta a. A seguir, para cada aresta $a \in \mathcal{L}$ da direita para a esquerda, percorre os vértices de a procurando por vértices u em a não visitados, faz $Visitado[u] = true$ e para o último vértice u não visitado faz $g[u] = (j - \sum_{v \in a \wedge Visitado[v]=true} g[v]) \bmod r$.

***Programa 5.37** Rotula grafo e atribui valores para o arranjo g*

```
Procedure Atribuig (var Grafo: TipoGrafo;
                    var L    : TipoArranjoArestas;
                    var g    : Tipog);
var i, j, u, Soma: integer;
    v: TipoValorVertice;
    a: TipoAresta;
    Visitado: array[0..MAXNUMVERTICES] of boolean;
begin
  for i := Grafo.NumVertices - 1 downto 0 do
    begin 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 + g[a.Vertices[v]];
    g[u] := (j - Soma) mod grafo.r;
    end;
end; { -Fim Atribuig- }
```