

0	1	2
0		9
1	7	8
2	7	9
3		8
4	7	
5	8	9

Figura 7.22 Representação por matrizes de incidência para o hipergrafo da Figura 7.21.

A representação por matrizes de incidência demanda muita memória para hipergrafos densos, em que  $|A|$  é próximo de  $|V|^2$ . Nessa representação, o tempo necessário para acessar um elemento é independente de  $|V|$  ou  $|A|$ . Logo, essa representação é muito útil para algoritmos em que necessitamos saber com rapidez se um vértice participa de determinada aresta. A maior desvantagem é que a matriz necessita  $\Omega(|V|^3)$  de espaço. Isso significa que simplesmente ler ou examinar a matriz tem complexidade de tempo  $O(|V|^3)$ .

O Programa 7.23 apresenta a estrutura de dados do **tipo abstrato de dados hipergrafo** utilizando matriz de incidência. O campo `Mat` é o principal componente do registro `TipoGrafo`, em que os itens são armazenados em um array de duas dimensões de tamanho suficiente para armazenar o grafo. As constantes `MaxNumVertices` e `MaxNumArestas` definem o maior número de vértices e de arestas que o grafo pode ter e `r` define o número de vértices de cada aresta.

Programa 7.23 Estrutura do tipo hipergrafo implementado como matriz de incidência

```
const MAXNUMVERTICES = 100;
      MAXNUMARESTAS = 4500;
      MAXR = 5;

type TipoValorVertice = 0..MAXNUMVERTICES;
    TipoValorAresta = 0..MAXNUMARESTAS;
    Tipor = 0..MAXR;
    TipoPesoAresta = integer;
    TipoArranjoVertices = array[Tipor] of TipoValorVertice;
    TipoAresta = record
        Vertices: TipoArranjoVertices;
        Peso : TipoPesoAresta;
    end;

    TipoGrafo = record
        Mat: array[TipoValorVertice, TipoValorAresta]
            of TipoPesoAresta;
        NumVertices : TipoValorVertice;
        NumArestas : TipoValorAresta;
        ProxDisponivel: TipoValorAresta;
        r : Tipor;
    end;
    TipoApontador = TipoValorAresta;
```

Uma possível implementação para as primeiras seis operações definidas anteriormente é mostrada no Programa 7.24. O procedimento `ArestasIguais` permite a comparação de duas arestas, a um custo  $O(r)$ . O procedimento `InserereAresta` tem custo  $O(r)$  e os procedimentos `ExisteAresta` e `RetiraAresta` têm custo  $r \times |A|$ , o que pode ser considerado  $O(|A|)$  porque  $r$  é geralmente uma constante pequena.

Programa 7.24 Operadores sobre hipergrafos implementados como matrizes de incidência

```
function ArestasIguais (var Vertices: TipoArranjoVertices;
                        NumAresta : TipoValorAresta;
                        var Grafo : TipoGrafo): boolean;
var Aux: boolean; v: Tipor;
begin
    Aux := true; v := 0;
    while (v < Grafo.r) and Aux do
        begin
            if Grafo.Mat[Vertices[v], NumAresta] <= 0 then Aux := false;
            v := v + 1;
        end;
    ArestasIguais := Aux;
end; { ArestasIguais }

procedure FGVazio (var Grafo: TipoGrafo);
var i, j: integer;
begin
    Grafo.ProxDisponivel := 0;
    for i := 0 to Grafo.NumVertices do
        for j := 0 to Grafo.NumArestas do Grafo.Mat[i, j] := 0;
    end; { FGVazio }

procedure InserereAresta (var Aresta: TipoAresta;
                          var Grafo : TipoGrafo);
var i: integer;
begin
    if Grafo.ProxDisponivel = MAXNUMARESTAS + 1
    then writeln ('Nao ha espaco disponivel para a aresta')
    else begin
        for i := 0 to Grafo.r - 1 do
            Grafo.Mat[Aresta.Vertices[i], Grafo.ProxDisponivel] := Aresta.Peso;
            Grafo.ProxDisponivel := Grafo.ProxDisponivel + 1;
        end;
    end; { InserereAresta }

function ExisteAresta (var Aresta: TipoAresta;
                      var Grafo : TipoGrafo): boolean;
var ArestaAtual: TipoValorAresta; EncontrouAresta: boolean;
begin
    EncontrouAresta := false; ArestaAtual := 0;
    while (ArestaAtual < Grafo.NumArestas) and not EncontrouAresta do
        begin
            if ArestasIguais(Aresta.Vertices, ArestaAtual, Grafo)
```