

Continuação do Programa 7.24

```

then EncontrouAresta := true;
ArestaAtual := ArestaAtual + 1;
end;
ExisteAresta := EncontrouAresta;
end; { ExisteAresta }

function RetiraAresta (var Aresta: TipoAresta;
var Grafo: TipoGrafo): TipoAresta;
var ArestaAtual: TipoValorAresta;
i: integer;
EncontrouAresta: boolean;
begin
  EncontrouAresta := false; ArestaAtual := 0;
  while (ArestaAtual < Grafo.NumArestas) and not EncontrouAresta do
    begin
      if ArestasIguais (Aresta.Vertices, ArestaAtual, Grafo)
      then begin
        EncontrouAresta := true;
        Aresta.Peso := Grafo.Mat[Aresta.Vertices[0], ArestaAtual];
        for i := 0 to Grafo.r - 1 do
          Grafo.Mat[Aresta.Vertices[i], ArestaAtual] := -1;
        end;
        ArestaAtual := ArestaAtual + 1;
      end;
    end;
  RetiraAresta := Aresta;
end; { RetiraAresta }

procedure ImprimeGrafo (var Grafo: TipoGrafo);
var i, j: integer;
begin
  write (' ');
  for i := 0 to Grafo.NumArestas-1 do write (i:3); writeLn;
  for i := 0 to Grafo.NumVertices-1 do
    begin
      write (i:3);
      for j := 0 to Grafo.NumArestas-1 do write (Grafo.mat[i,j]:3);
      writeLn;
    end;
  end; { ImprimeGrafo }

function ListaIncVazia (var Vertice: TipoValorVertice;
var ArestaAtual: TipoArestador; ListaVazia: boolean;
begin
  ListaVazia := true; ArestaAtual := 0;

```

Continuação do Programa 7.24

```

while (ArestaAtual < Grafo.NumArestas) and ListaVazia do
  if Grafo.Mat[Vertice, ArestaAtual] > 0
  then ListaVazia := false
  else ArestaAtual := ArestaAtual + 1;
ListaIncVazia := ListaVazia = true;
end; { ListaIncVazia }

function PrimeiroListaInc (var Vertice: TipoValorVertice;
var ArestaAtual: TipoArestador; ListaVazia: boolean;
begin
  ListaVazia := true; ArestaAtual := 0;
  while (ArestaAtual < Grafo.NumArestas) and ListaVazia do
    if Grafo.Mat[Vertice, ArestaAtual] > 0
    then begin PrimeiroListaInc := ArestaAtual; ListaVazia := false; end
    else ArestaAtual := ArestaAtual + 1;
  if ArestaAtual = Grafo.NumArestas
  then writeLn ('Erro: Lista incidencia vazia');
end; { PrimeiroListaInc }

procedure ProxArestaInc (var Vertice: TipoValorVertice;
var Grafo: TipoGrafo;
var Inc: TipoValorAresta;
var Peso: TipoPesoAresta;
var Prox: TipoArestador;
var FimListaAdj: boolean);
{--Retorna proxima aresta Inc apontada por Prox--}
begin
  Inc := Prox; Peso := Grafo.Mat[Vertice, Prox]; Prox := Prox + 1;
  while (Prox < Grafo.NumArestas) and (Grafo.Mat[Vertice, Prox] = 0) do
    Prox := Prox + 1;
  FimListaAdj := (Prox = Grafo.NumArestas);
end; { ProxArestaInc }

```

7.10.2 Implementação por meio de Listas de Incidência Usando Arranjos

A estrutura de dados usada para representar um hipergrafo $G_r = (V, A)$ por meio de listas de incidência foi proposta por Ebert (1987). A estrutura usa arranjos para armazenar as arestas e as listas de arestas incidentes a cada vértice. A Figura 7.23(a) mostra o mesmo 3-grafo de 6 vértices e 3 arestas da Figura 7.21, e a Figura 7.23(b), a sua representação por listas de incidência.

As arestas são armazenadas em um arranjo chamado *Arestas*. Em cada posição a do arranjo *Arestas*, são armazenados os r vértices da aresta a e o seu *Peso*. As listas de arestas incidentes nos vértices do hipergrafo são armazenadas nos arranjos *Prim* e *Prox*. O elemento *Prim*[v] define o ponto de entrada para a lista de arestas incidentes no vértice v , enquanto *Prox*[*Prim*[v]], *Prox*[*Prox*[*Prim*[v]]] e