Continuação do Programa 7.3

```
procedure ProxAdj (Vertice        : TipoValorVertice;
                   var Grafo      : TipoGrafo;
                   var Adj        : TipoValorVertice;
                   var Peso       : TipoPeso;
                   var Prox       : TipoApontador;
                   var FimListaAdj : boolean);
{--Retorna Adj apontado por Prox--}
begin
  Adj := Prox; Peso := Grafo.Mat[Vertice, Prox]; Prox := Prox + 1;
  while (Prox < Grafo.NumVertices) and (Grafo.Mat[Vertice,Prox] = 0) do
    Prox := Prox + 1;
  if Prox = Grafo.NumVertices then FimListaAdj := true;
end; { ProxAdj }

procedure RetiraAresta (V1, V2: TipoValorVertice;
                        var Peso: TipoPeso; var Grafo: TipoGrafo);
begin
  if Grafo.Mat[V1, V2] = 0
  then writeln ('Aresta nao existe')
  else begin Peso := Grafo.Mat[V1, V2]; Grafo.Mat[V1, V2] := 0; end;
end; { RetiraAresta }

procedure LiberaGrafo (var Grafo: TipoGrafo);
begin
  { Nao faz nada no caso de matrizes de adjacencia }
end; { LiberaGrafo }

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

7.2.2 Implementação por meio de Listas de Adjacência Usando Apontadores

A representação de um grafo $G = (V, A)$ por **listas de adjacência** consiste de um arranjo Adj de $|V|$ listas, uma para cada vértice em V. Para cada $u \in V$,

a lista de adjacentes $Adj[u]$ contém todos os vértices v tal que existe uma aresta $(u, v) \in A$, isto é, $Adj[u]$ contém todos os vértices adjacentes a u em G. Os vértices de uma lista de adjacência são em geral armazenados em uma ordem arbitrária. A representação por listas de adjacências possui uma complexidade de espaço $O(|V|+|A|)$, sendo pois indicada para grafos **esparsos**, em que $|A|$ é muito menor do que $|V|^2$. Essa representação é compacta e geralmente utilizada na maioria das aplicações. Entretanto, a principal desvantagem dessa representação é que ela pode ter tempo $O(|V|)$ para determinar se existe uma aresta entre o vértice i e o vértice j, uma vez que podem existir $O(|V|)$ vértices na lista de adjacentes do vértice i.

A implementação de listas de adjacências pode ser realizada por meio das duas estruturas de dados usuais para representar listas lineares: apontadores e posições contíguas de memória. Esta seção apresenta a implementação de listas de adjacência usando apontadores, e a próxima apresenta a implementação de listas de adjacência usando posições contíguas de memória mediante arranjos.

As Figuras 7.8(a) e 7.8(b) apresentam a representação para listas de adjacência usando apontadores para um grafo direcionado contendo quatro vértices e três arestas e para um grafo não direcionado contendo quatro vértices e duas arestas, respectivamente. Note que cada aresta é representada duas vezes no grafo não direcionado.

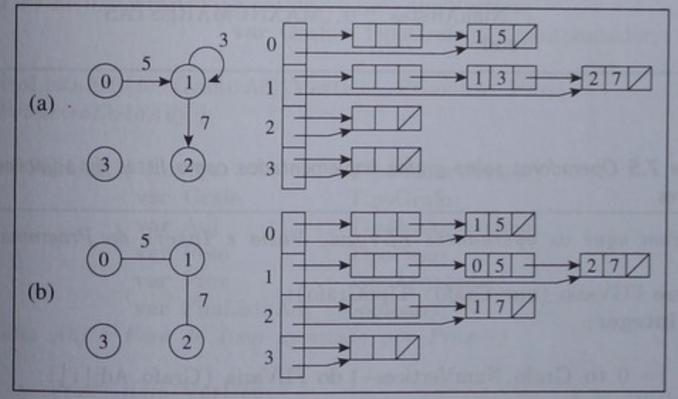

Figura 7.8 *Representação para listas de adjacência usando apontadores. (a) Grafo direcionado; (b) Grafo não direcionado.*

No uso de apontadores, a lista é constituída de células, em que cada célula contém um item da lista e um apontador para a célula seguinte. O registro TipoLista contém um apontador para a célula cabeça e um apontador para a última célula da lista, conforme mostra o Programa 7.4.

Uma possível implementação para as operações definidas anteriormente é mostrada no Programa 7.5.