

Programa 7.6 Estrutura do tipo grafo implementado como listas de adjacência usando arranjos

```

const MAXNUMVERTICES = 100;
      MAXNUMARESTAS  = 4500;
      MAXTAM         = MAXNUMVERTICES + 2 * MAXNUMARESTAS;

type
  TipoValorVertice = 0..MAXNUMVERTICES;
  TipoPeso         = integer;
  TipoTam          = 0..MAXTAM;
  TipoGrafo        = record
    Cab      : array[TipoTam] of TipoTam;
    Prox      : array[TipoTam] of TipoTam;
    Peso      : array[TipoTam] of TipoTam;
    ProxDisponivel: TipoTam;
    NumVertices : 0..MAXNUMVERTICES;
    NumArestas  : 0..MAXNUMARESTAS;
  end;
  TipoApontador = TipoTam;

```

Programa 7.7 Operadores implementados como listas de adjacência usando arranjos

```

procedure FGVazio(var Grafo: TipoGrafo);
var i: integer;
begin
  for i := 0 to Grafo.NumVertices do
    begin
      Grafo.Prox[i] := 0; Grafo.Cab[i] := i;
      Grafo.ProxDisponivel := Grafo.NumVertices;
    end;
end;

procedure InsereAresta (V1, V2: TipoValorVertice;
                       Peso : TipoPeso;
                       var Grafo : TipoGrafo);
var Pos: integer;
begin
  Pos := Grafo.ProxDisponivel;
  if Grafo.ProxDisponivel = MAXTAM
  then writeln('nao ha espaco disponivel para a aresta')
  else begin
    Grafo.ProxDisponivel := Grafo.ProxDisponivel + 1;
    Grafo.Prox[Grafo.Cab[V1]] := Pos;
    Grafo.Cab[Pos] := V2; Grafo.Cab[V1] := Pos;
    Grafo.Prox[Pos] := 0; Grafo.Peso[Pos] := Peso;
  end;
end; {InsereAresta}

```

Continuação do Programa 7.7

```

function ExisteAresta (Vertice1, Vertice2: TipoValorVertice;
                      var Grafo: TipoGrafo): boolean;
var Aux: TipoApontador; EncontrouAresta: boolean;
begin
  Aux := Grafo.Prox[Vertice1]; EncontrouAresta := false;
  while (Aux <> 0) and (EncontrouAresta = false) do
    begin
      if Vertice2 = Grafo.Cab[Aux] then EncontrouAresta := true;
      Aux := Grafo.Prox[Aux];
    end;
  ExisteAresta := EncontrouAresta;
end; { ExisteAresta }

{-- Operadores para obter a lista de adjacentes --}
function ListaAdjVazia (Vertice: TipoValorVertice;
                       var Grafo: TipoGrafo): boolean;
begin
  ListaAdjVazia := Grafo.Prox[Vertice] = 0; end;

function PrimeiroListaAdj (Vertice: TipoValorVertice;
                          var Grafo: TipoGrafo): TipoApontador;
begin
  PrimeiroListaAdj := Grafo.Prox[Vertice]; end;

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 := Grafo.Cab[Prox];
  Peso := Grafo.Peso[Prox];
  Prox := Grafo.Prox[Prox];
  if Prox = 0 then FimListaAdj := true;
end; { ProxAdj }

procedure RetiraAresta (V1, V2: TipoValorVertice;
                       var Peso: TipoPeso; var Grafo: TipoGrafo);
var Aux, AuxAnterior: TipoApontador; EncontrouAresta: boolean;
begin
  AuxAnterior := V1; Aux := Grafo.Prox[V1]; EncontrouAresta := false;
  while (Aux <> 0) and (EncontrouAresta = false) do
    begin
      if V2 = Grafo.Cab[Aux]
      then EncontrouAresta := true
      else begin AuxAnterior := Aux; Aux := Grafo.Prox[Aux]; end;
    end;
  if EncontrouAresta {-- Apenas marca como retirado --}
  then Grafo.Cab[Aux] := MAXNUMVERTICES + 2 * MAXNUMARESTAS
  else writeln('Aresta nao existe');
end; { RetiraAresta }

```