

Continuação do Programa 7.10

```

begin
  NArestas := Grafo.NumArestas; FFVazia (Fila); j := 0;
  while j < Grafo.NumVertices do
    begin
      if VerticeGrauUm (j, Grafo)
      then begin x.Chave := j; Enfileira (x, Fila); end;
      j := j + 1;
    end;
  while not Vazia (Fila) and (NArestas > 0) do
    begin
      Desenfileira (Fila, x);
      if Grafo.Prim[x.Chave] >= 0
      then begin
        A1 := Grafo.Prim[x.Chave] mod Grafo.NumArestas;
        Aresta := RetiraAresta (Grafo.Arestas[A1], Grafo);
        L[Grafo.NumArestas - NArestas] := Aresta;
        NArestas := NArestas - 1;
        if NArestas > 0
        then for j := 0 to Grafo.r - 1 do
          if VerticeGrauUm (Aresta.Vertices[j], Grafo)
          then begin
            x.Chave := Aresta.Vertices[j];
            Enfileira (x, Fila);
          end;
        end;
      end;
    end;
  { else writeln ('Nao ha vertices de grau 1 no grafo'); }
  GAciclico := NArestas = 0;
end; { GrafoAciclico }

```

7.5 Busca em Largura

A busca em largura (do inglês *breadth-first search*) é assim chamada porque ela expande a fronteira entre vértices descobertos e não descobertos uniformemente por meio da largura da fronteira, como se fossem círculos concêntricos gerados por uma pedra que se deixa cair em uma superfície de água completamente parada. O algoritmo é a base para muitos algoritmos em grafos importantes, tais como o algoritmo de Prim para obter a árvore geradora mínima (Seção 7.8.2) e o algoritmo de Dijkstra para obter o caminho mais curto de um vértice a todos os outros vértices (Seção 7.9). Dados um grafo $G = (V, A)$ e um vértice origem, o algoritmo de busca em largura descobre todos os vértices a uma distância k do vértice origem antes de descobrir qualquer vértice a uma distância $k + 1$. O grafo $G = (V, A)$ pode ser direcionado ou não direcionado.

Dado um grafo $G(V, A)$ e um vértice origem u , a busca em largura explora sistematicamente as arestas de G com o objetivo de descobrir todos os vértices que são alcançáveis a partir de u . Para acompanhar o progresso do algoritmo, cada vértice é colorido de branco, cinza ou preto. Todos os vértices são inicializados brancos, podem posteriormente se tornar cinza e, finalmente, pretos. Quando um vértice é descoberto pela primeira vez durante a busca, ele se torna cinza. Assim, vértices cinza e pretos já foram descobertos, mas a busca em largura distingue entre eles para assegurar que a busca ocorra em largura. Se $(u, v) \in A$ e o vértice u é preto, então o vértice v tem de ser cinza ou preto, o que significa que todos os vértices adjacentes a vértices pretos já foram descobertos. Vértices cinza podem ter alguns vértices adjacentes brancos, e eles representam a fronteira entre vértices descobertos e não descobertos.

O Programa 7.11 implementa a busca em largura. O algoritmo `VisitaBfs` obtém o menor número de arestas entre o vértice origem u e todo vértice que possa ser alcançado. O grafo de entrada G pode ser direcionado ou não direcionado. O algoritmo usa uma fila do tipo “primeiro-que-chega, primeiro-atendido” para gerenciar o conjunto de vértices cinza.

Programa 7.11 Busca em largura

```

{— Entram aqui os operadores FFVazia, Vazia, Enfileira e —}
{— Desenfileira do Programa 3.18 ou do Programa 3.20, —}
{— dependendo da implementação da busca em largura usar —}
{— arranjos ou apontadores, respectivamente —}

```

```

procedure BuscaEmLargura (var Grafo: TipoGrafo);
var x      : TipoValorVertice;
    Dist   : array[TipoValorVertice] of integer;
    Cor     : array[TipoValorVertice] of TipoCor;
    Antecessor : array[TipoValorVertice] of integer;
procedure VisitaBfs (u: TipoValorVertice);
var v      : TipoValorVertice;
    Aux     : TipoApontador;
    FimListaAdj: boolean;
    Peso     : TipoPeso;
    Item     : TipoItem;
    Fila     : TipoFila;
begin
  Cor[u] := cinza;
  Dist[u] := 0;
  FFVazia (Fila);
  Item.Vertice := u;
  Enfileira (Item, Fila);
  write ('Visita origem', u:2, ' cor: cinza F:');
  ImprimeFila (Fila); readln;
  while not FilaVazia (Fila) do
    begin
      Desenfileira (Fila, Item); u := Item.vertice;

```