representam conexões diretas entre elas. O peso de cada aresta representa o tempo gasto para transferir uma mensagem de tamanho M entre duas máquinas diretamente conectadas. Devemos utilizar a **árvore de caminhos mais curtos** do servidor a todas as outras máquinas.

d) Devemos utilizar o **algoritmo de Dijkstra**, utilizando como origem a máquina servidora. O algoritmo de Dijkstra sempre produz soluções ótimas pois é um algoritmo guloso num contexto de subestrutura ótima. Sua complexidade é $O(|A|log(|V|))$, sendo$|V|$ o número de vértices e $|A|$ o de arestas.

e) Como todas as arestas têm peso igual, a árvore de caminhos mais curtos pode ser determinada usando **busca em largura**, cuja ordem de complexidade é $O(|V| + |A|)$.

7.4.

Programa K.16 *Classificação de arestas*

```
program BuscaEmProfundidadeListaApClassificaArestas;
{-- Implementacao TAD Grafo com listas/apontadores      --}
{-- Imprime floresta de arvores de busca em profundidade --}
const MaxNumvertices = 100;
      MaxNumArestas  = 100;
type TipoValorTempo = 0..2*MaxNumvertices;
     TipoCor = (branco, cinza, preto);
{-- Entram aqui os tipos do Programa 7.4 --}
var i           : integer;
    V1, V2, Adj: TipoValorVertice;
    A           : TipoValorAresta;
    Grafo       : TipoGrafo;
    x           : TipoItem;
    FimListaAdj: boolean;
    NVertices   : TipoValorVertice;
    NArestas    : TipoValorAresta;
{-- Entram aqui os operadores do Programa 3.4 --}
{-- Entram aqui os operadores do Programa 7.5 --}
procedure BuscaEmProfundidade (var Grafo: TipoGrafo);
var Tempo       : TipoValorTempo;
    x           : TipoValorVertice;
    d, t        : array[TipoValorVertice] of TipoValorTempo;
    Cor         : array[TipoValorVertice] of TipoCor;
    Antecessor : array[TipoValorVertice] of integer;
  procedure VisitaDfs (u:TipoValorVertice);
  var FimListaAdj: boolean;
      Peso       : TipoValorAresta;
      Aux        : Apontador;
      v          : TipoValorVertice;
  begin
    Cor[u] := cinza;
    Tempo := Tempo + 1;  d[u] := Tempo;
    writeln('Visita',u:2,' Tempo descoberta:',d[u]:2,' cinza'); readln;
```


Continuação do Programa K.16

```
    if not ListaAdjVazia (u, Grafo)
    then begin
         Aux := PrimeiroListaAdj (u, Grafo); FimListaAdj := false;
         while not FimListaAdj do
           begin
           ProxAdj (u, v, Peso, Aux, FimListaAdj);
           if Cor[v] = branco
           then begin
                writeln ('Aresta arvore:',u:2,'->',v:2,' (branco)');
                Antecessor[v] := u;  VisitaDfs (v);
                end
           else if cor[v] = cinza
                then writeln ('Aresta de retorno:',
                              u:2,'->',v:2,' (cinza)')
                else if d[u] > d[v]
                     then writeln ('Aresta decruzamento:',
                                   u:2,'->',v:2,' (preto)')
                     else writeln ('Aresta de avanco:',
                                   u:2,'->',v:2,' (preto)');
           end;
         end;
    Cor[u] := preto;  Tempo := Tempo + 1;  t[u] := Tempo;
    writeln ('Visita',u:2,' Tempo termino:',t[u]:2,' preto'); readln;
  end;
begin
  Tempo := 0;
  for x := 0 to Grafo.NumVertices-1 do
    begin  Cor[x] := branco;  Antecessor[x] := -1;  end;
  for x := 0 to Grafo.Numvertices-1 do
    if Cor[x] = branco
    then begin
         writeln ('Raiz arvore:',x:2,' (branco)');  VisitaDfs (x);
         end;
end;
begin {-- Programa principal --}
  write ('No. vertices:'); readln (NVertices);
  write ('No. arestas:'); readln (NArestas);
  Grafo.NumVertices := NVertices;  Grafo.NumArestas := 0;
  FGVazio (Grafo);
  for i := 0 to NArestas-1 do
    begin
    write ('Insere V1 -- V2 -- Aresta:');  readln (V1, V2, A);
    Grafo.NumArestas := Grafo.NumArestas + 1;
    InsereAresta (V1, V2, A, Grafo); {1 chamada: G direcionado}
    {InsereAresta (V2, V1, A, Grafo);} {2 chamadas: G nao-direcionado}
    end;
  ImprimeGrafo (Grafo);  readln;
  BuscaEmProfundidade (Grafo);  readln;
end.
```