vértice restante do exemplo). Cada árvore nessa floresta forma um componente fortemente conectado.

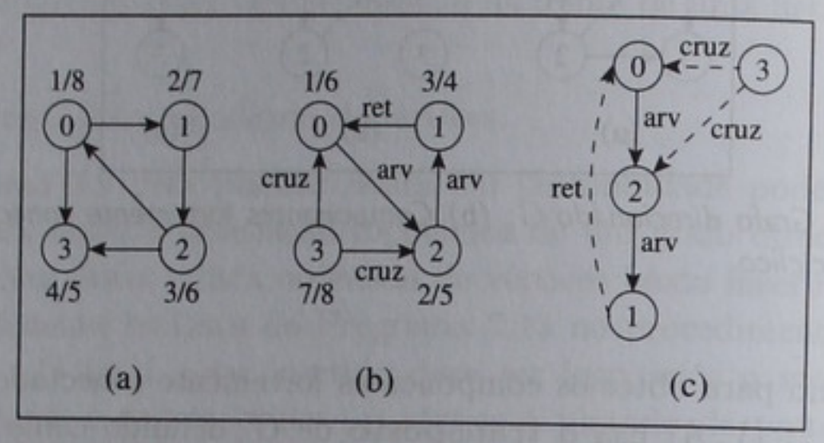

Figura 7.15 *(a) Grafo direcionado G; (b) Grafo transposto G^T; (c) Floresta constituída de duas árvores de busca em profundidade.*

Dado um grafo direcionado $G = (u, v)$, o Programa 7.14 obtém o grafo transposto G^T. A implementação para obter o grafo transposto, como nos casos anteriores, é independente da estrutura de dados utilizada. Qualquer uma das três implementações apresentadas nas Seções 7.2.1, 7.2.2 ou 7.2.3 pode ser usada com o programa.

Programa 7.14 *Obtém o grafo transposto G^T a partir de um grafo G*

```
procedure GrafoTransposto (var Grafo, var GrafoT: TipoGrafo);
var v, Adj: TipoValorVertice;  i: integer;
    Peso: TipoPeso;  Aux: TipoApontador;
begin
  FGVazio (GrafoT);
  GrafoT.NumVertices := Grafo.NumVertices;
  GrafoT.NumArestas  := Grafo.NumArestas;
  for i := 0 to Grafo.NumVertices-1 do
    begin
    v := i;
    if not ListaAdjVazia (v, Grafo)
    then begin
           Aux := PrimeiroListaAdj (v, Grafo);  FimListaAdj := false;
           while not FimListaAdj do
             begin
             ProxAdj (v, Grafo, Adj, Peso, Aux, FimListaAdj);
             InsereAresta (Adj, v, Peso, GrafoT);
             end;
         end;
    end;
end; { GrafoTransposto }
```


Depois de obter os tempos de término $t[u]$ utilizando o Programa 7.9 e obter G^T a partir de G com o emprego do Programa 7.14, o passo seguinte é realizar uma busca em profundidade em G^T a partir do vértice de maior $t[u]$. No Programa 7.15, a ordem de processamento dos vértices tem início a partir do vértice de maior tempo de término e prossegue na ordem decrescente de tempo de busca enquanto existirem vértices remanescentes.

Programa 7.15 *Busca em profundidade no grafo transposto G^T*

```
procedure BuscaEmProfundidadeCfc (var Grafo: TipoGrafo;
                                  var TT   : TipoTempoTermino);
var
  Tempo      : TipoValorTempo;
  x, VRaiz   : 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: TipoPeso;
      Aux: TipoApontador;  v: TipoValorVertice;
  begin
    Cor[u] := cinza;  Tempo := Tempo + 1;  d[u] := Tempo;
    TT.Restantes[u] := false;  TT.NumRestantes := TT.NumRestantes-1;
    writeln('Visita',u:2,' Tempo descoberta:',d[u]:2,' cinza'); readln;
    if not ListaAdjVazia (u, Grafo)
    then begin
           Aux := PrimeiroListaAdj (u, Grafo);  FimListaAdj := false;
           while not FimListaAdj do
             begin
             ProxAdj (u, Grafo, v, Peso, Aux, FimListaAdj);
             if Cor[v] = branco
             then begin Antecessor[v] := u; VisitaDfs (v); end;
             end;
         end;
    Cor[u] := preto;  Tempo := Tempo + 1;  t[u] := Tempo;
    writeln ('Visita',u:2,' Tempo termino:',t[u]:2,' preto'); readln;
  end; { VisitaDfs }

begin
  Tempo := 0;
  for x := 0 to Grafo.NumVertices-1 do
    begin Cor[x] := branco;  Antecessor[x] := -1; end;
  TT.NumRestantes := Grafo.NumVertices;
  for x := 0 to Grafo.NumVertices-1 do TT.Restantes[x] := true;
  while TT.NumRestantes > 0 do
    begin
    VRaiz := MaxTT (TT); writeln('Raiz da proxima arvore:',VRaiz:2);
    VisitaDfs (VRaiz);
    end;
end; { BuscaEmProfundidadeCfc }
```