

Programa G.10 Teste para verificar se um hipergrafo é acíclico usando arranjos

```

{-- Entram aqui os tipos do Programa C.17 (ou do Programa C.19) --}
{-- Entram aqui tipos do Programa G.25 --}
int i, j;
TipoAresta Aresta;
TipoGrafo Grafo;
TipoArranjoArestas L;
short GAciclico;
{-- Entram aqui os operadores FFVazia, Vazia, Enfileira e --}
{-- Desenfileira do Programa C.18 (ou do Programa C.20) --}
{-- Entram aqui os operadores ArestasIguais, FGVazio, --}
{-- InserirAresta, RetiraAresta e ImprimeGrafo do Programa G.26 --}

short VerticeGrauUm(TipoValorVertice *V,
                    TipoGrafo *Grafo)
{
    return (Grafo->Prim[*V] >= 0) &&
           (Grafo->Prox[Grafo->Prim[*V]] == INDEFINIDO);
}

void GrafoAciclico (TipoGrafo *Grafo,
                    TipoArranjoArestas L, short *GAciclico)
{
    TipoValorVertice j = 0; TipoValorAresta A1;
    Tipoltem x; TipoFila Fila; TipoValorAresta NArestas;
    TipoAresta Aresta; NArestas = Grafo->NumArestas;
    FFVazia (&Fila);
    while (j < Grafo->NumVertices)
    {
        if (VerticeGrauUm (&j, Grafo))
        {
            x.Chave = j; Enfileira (x, &Fila);
            j++;
        }
        while (!Vazia(&Fila) && (NArestas > 0))
        {
            Desenfileira (&Fila, &x);
            if (Grafo->Prim[x.Chave] >= 0)
            {
                A1 = Grafo->Prim[x.Chave] % Grafo->NumArestas;
                Aresta = RetiraAresta(&Grafo->Arestas[A1], Grafo);
                L[Grafo->NumArestas - NArestas] = Aresta;
                NArestas = NArestas - 1;
                if (NArestas > 0)
                {
                    for (j = 0; j < Grafo->r; j++)
                    {
                        if (VerticeGrauUm(&Aresta.Vertices[j], Grafo))
                        {
                            x.Chave = Aresta.Vertices[j]; Enfileira (x, &Fila);
                        }
                    }
                }
            }
        }
        if (NArestas == 0) *GAciclico = TRUE;
        else *GAciclico = FALSE;
    }
}

```

Programa G.11 Busca em largura

```

{-- Entram aqui os operadores FFVazia, Vazia, Enfileira e --}
{-- Desenfileira do Programa C.18 ou do Programa C.20 --}
{-- dependendo da implementação da busca em largura usar --}
{-- arranjos ou apontadores, respectivamente --}

void VisitaBfs(TipoValorVertice u, TipoGrafo *Grafo,
               int *Dist, TipoCor *Cor, int *Antecessor)
{
    TipoValorVertice v; Apontador Aux; short FimListaAdj;
    TipoPeso Peso; Tipoltem Item; TipoFila Fila;
    Cor[u] = cinza; Dist[u] = 0;
    FFVazia(&Fila);
    Item.Vertice = u; Item.Peso = 0;
    Enfileira (Item, &Fila);
    printf("Visita origem %2d cor: cinza F:", u);
    ImprimeFila(Fila); getchar();
    while (!FilaVazia(Fila))
    {
        Desenfileira(&Fila, &Item);
        u = Item.Vertice;
        if (!ListaAdjVazia(&u, Grafo))
        {
            Aux = PrimeiroListaAdj(&u, Grafo);
            FimListaAdj = FALSE;
            while (FimListaAdj == FALSE)
            {
                ProxAdj(&u, &v, &Peso, &Aux, &FimListaAdj);
                if (Cor[v] != branco) continue;
                Cor[v] = cinza; Dist[v] = Dist[u] + 1;
                Antecessor[v] = u;
                Item.Vertice = v; Item.Peso = Peso;
                Enfileira (Item, &Fila);
            }
        }
        Cor[u] = preto;
        printf("Visita %2d Dist %2d cor: preto F:", u, Dist[u]);
        ImprimeFila(Fila); getchar();
    }
} /* VisitaBfs */

void BuscaEmLargura(TipoGrafo *Grafo)
{
    TipoValorVertice x;
    int Dist[MaxNumVertices + 1];
    TipoCor Cor[MaxNumVertices + 1];
    int Antecessor[MaxNumVertices + 1];
    for (x = 0; x <= Grafo->NumVertices - 1; x++)
    {
        Cor[x] = branco; Dist[x] = Infinito; Antecessor[x] = -1;
    }
    for (x = 0; x <= Grafo->NumVertices - 1; x++)
    {
        if (Cor[x] == branco)
            VisitaBfs (x, Grafo, Dist, Cor, Antecessor);
    }
}

```