

Programa G.12 Imprime os vértices do caminho mais curto entre o vértice origem e outro vértice qualquer do grafo

```
void ImprimeCaminho(TipoValorVertice Origem, TipoValorVertice v,
                   TipoGrafo *Grafo, int *Dist, TipoCor *Cor,
                   int *Antecessor)
{
    if (Origem == v)
    {
        printf("%d ", Origem);
        return;
    }
    if (Antecessor[v] == -1)
        printf("Nao existe caminho de %d ate %d", Origem, v);
    else { ImprimeCaminho(Origem, Antecessor[v],
                          Grafo, Dist, Cor, Antecessor);
          printf("%d ", v);
        }
}
```

Programa G.13 Insere em uma lista encadeada antes do primeiro item da lista

```
void InsLista (TipoItem *Item, TipoLista *Lista)
{
    /*— Insere antes do primeiro item da lista —*/
    TipoApontador Aux;
    Aux = Lista->Primeiro->Prox;
    Lista->Primeiro->Prox = (TipoApontador)malloc(sizeof(tipoCelula));
    Lista->Primeiro->Prox->Item = Item;
    Lista->Primeiro->Prox->Prox = Aux;
}
```

Programa G.14 Obtém o grafo transposto G^T a partir de um grafo G

```
void GrafoTransposto(TipoGrafo *Grafo, TipoGrafo *GrafoT)
{
    TipoValorVertice v, Adj;
    TipoPeso Peso;
    TipoApontador Aux;
    FGVazio(GrafoT);
    GrafoT->NumVertices = Grafo->NumVertices;
    GrafoT->NumArestas = Grafo->NumArestas;
    for (v = 0; v <= Grafo->NumVertices - 1; v++)
    {
        if (!ListaAdjVazia(&v, Grafo))
        {
            Aux = PrimeiroListaAdj(&v, Grafo);
            FimListaAdj = FALSE;
            while (!FimListaAdj)
            {
                ProxAdj(&v, Grafo, &Adj, &Peso, &Aux, &FimListaAdj);
                InsereAresta(&Adj, &v, &Peso, GrafoT);
            }
        }
    }
}
```

Programa G.15 Busca em profundidade no grafo transposto G^T

```
void VisitaDfs2(TipoValorVertice u, TipoGrafo *Grafo,
               TipoTempoTermino *TT, TipoValorTempo *Tempo,
               TipoValorTempo *d, TipoValorTempo *t,
               TipoCor *Cor, short *Antecessor)
{
    short FimListaAdj;
    TipoPeso Peso;
    TipoApontador Aux;
    TipoValorVertice v;
    Cor[u] = cinza;
    (*Tempo)++; d[u] = (*Tempo);
    TT->Restantes[u] = FALSE;
    TT->NumRestantes--;
    printf("Visita %2d Tempo descoberta: %2d cinza\n", u, d[u]);
    getchar();
    if (!ListaAdjVazia(&u, Grafo))
    {
        Aux = PrimeiroListaAdj(&u, Grafo);
        FimListaAdj = FALSE;
        while (!FimListaAdj)
        {
            ProxAdj(&u, Grafo, &v, &Peso, &Aux, &FimListaAdj);
            if (Cor[v] == branco)
            {
                Antecessor[v] = u;
                VisitaDfs2(v, Grafo, TT, Tempo, d, t, Cor, Antecessor);
            }
        }
    }
    Cor[u] = preto; (*Tempo)++;
    t[u] = (*Tempo);
    printf("Visita %2d Tempo termino: %2d preto\n", u, t[u]);
    getchar();
}

void BuscaEmProfundidadeCfc(TipoGrafo *Grafo, TipoTempoTermino *TT)
{
    TipoValorTempo Tempo;
    TipoValorTempo d[MAXNUMVERTICES + 1], t[MAXNUMVERTICES + 1];
    TipoCor Cor[MAXNUMVERTICES + 1];
    short Antecessor[MAXNUMVERTICES + 1];
    TipoValorVertice x, VRaiz; Tempo = 0;
    for (x = 0; x <= Grafo->NumVertices - 1; x++)
    {
        Cor[x] = branco; Antecessor[x] = -1;
    }
    TT->NumRestantes = Grafo->NumVertices;
    for (x = 0; x <= Grafo->NumVertices - 1; x++)
    {
        TT->Restantes[x] = TRUE;
        while (TT->NumRestantes > 0)
        {
            VRaiz = MaxTT(TT, Grafo);
            printf("Raiz da proxima arvore: %2d\n", VRaiz);
            VisitaDfs2(VRaiz, Grafo, TT, &Tempo, d, t, Cor, Antecessor);
        }
    }
}
```