

Continuação do Programa G.23

```

typedef int TipoValorAresta;
typedef int Tipor;
typedef int TipoPesoAresta;
typedef TipoValorVertice TipoArranjoVertices [MAXR];
typedef struct TipoAresta {
    TipoArranjoVertices Vertices;
    TipoPesoAresta Peso;
} TipoAresta;
typedef struct TipoGrafo {
    TipoPesoAresta Mat[MAXNUMVERTICES][MAXNUMARESTAS];
    TipoValorVertice NumVertices;
    TipoValorAresta NumArestas;
    TipoValorAresta ProxDisponivel;
    Tipor r;
} TipoGrafo;
typedef TipoValorAresta TipoApontador;

```

Programa G.24 Operadores sobre hipergrafos implementados como matrizes de incidência

```

short ArestasIguais (TipoArranjoVertices * Vertices,
                    TipoValorAresta NumAresta,
                    TipoGrafo * Grafo)
{
    short Aux = TRUE;
    while (v < Grafo->r && Aux == TRUE)
    {
        if (Grafo->Mat[( * Vertices )][v][NumAresta] <= 0) Aux = FALSE;
        v = v + 1;
    }
    return Aux;
}

void FGVazio (TipoGrafo * Grafo)
{
    int i, j;
    Grafo->ProxDisponivel = 0;
    for (i = 0; i < Grafo->NumVertices; i++)
        for (j = 0; j < Grafo->NumArestas; j++) Grafo->Mat[i][j] = 0;
}

void InsereAresta (TipoAresta * Aresta, TipoGrafo * Grafo)
{
    int i;
    if (Grafo->ProxDisponivel == MAXNUMARESTAS)
        printf("Nao ha espaco disponivel para a aresta\n");
    else
    {
        for (i = 0; i < Grafo->r; i++)
            Grafo->Mat[Aresta->Vertices[i]][Grafo->ProxDisponivel] = Aresta->Peso;
        Grafo->ProxDisponivel = Grafo->ProxDisponivel + 1;
    }
}

```

Continuação do Programa G.24

```

short ExisteAresta (TipoAresta * Aresta, TipoGrafo * Grafo)
{
    TipoValorAresta ArestaAtual = 0;
    short EncontrouAresta = FALSE;
    while (ArestaAtual < Grafo->NumArestas &&
           EncontrouAresta == FALSE)
    {
        EncontrouAresta =
            ArestasIguais(&(Aresta->Vertices), ArestaAtual, Grafo);
        ArestaAtual = ArestaAtual + 1;
    }
    return EncontrouAresta;
}

TipoAresta RetiraAresta (TipoAresta * Aresta, TipoGrafo * Grafo)
{
    TipoValorAresta ArestaAtual = 0;
    int i;
    short EncontrouAresta = FALSE;
    while (ArestaAtual < Grafo->NumArestas & EncontrouAresta == FALSE)
    {
        if (ArestasIguais(&(Aresta->Vertices), ArestaAtual, Grafo))
        {
            EncontrouAresta = TRUE;
            Aresta->Peso = Grafo->Mat[Aresta->Vertices[0]][ArestaAtual];
            for (i = 0; i < Grafo->r; i++)
                Grafo->Mat[Aresta->Vertices[i]][ArestaAtual] = -1;
        }
        ArestaAtual = ArestaAtual + 1;
    }
    return *Aresta;
}

void ImprimeGrafo (TipoGrafo * Grafo)
{
    int i, j;
    printf("\n");
    for (i = 0; i < Grafo->NumArestas; i++) printf("%3d", i);
    printf("\n");
    for (i = 0; i < Grafo->NumVertices; i++)
    {
        printf("%3d", i);
        for (j = 0; j < Grafo->NumArestas; j++)
            printf("%3d", Grafo->Mat[i][j]);
        printf("\n");
    }
}

short ListaIncVazia (TipoValorVertice * Vertice, TipoGrafo * Grafo)
{
    short ListaVazia = TRUE;
    TipoApontador ArestaAtual = 0;
    while (ArestaAtual < Grafo->NumArestas && ListaVazia == TRUE)
    {
        if (Grafo->Mat[*Vertice][ArestaAtual] > 0)
            ListaVazia = FALSE;
        else ArestaAtual = ArestaAtual + 1;
    }
    return ListaVazia;
}

```