Programa G.3 *Operadores sobre grafos implementados como matrizes de adjacência*

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

void InsereAresta(TipoValorVertice *V1, TipoValorVertice *V2,
                  TipoPeso *Peso, TipoGrafo *Grafo)
{ Grafo->Mat[*V1][*V2] = *Peso; }

short ExisteAresta(TipoValorVertice Vertice1,
                   TipoValorVertice Vertice2, TipoGrafo *Grafo)
{ return (Grafo->Mat[Vertice1][Vertice2] > 0); }

/* Operadores para obter a lista de adjacentes */
short ListaAdjVazia(TipoValorVertice *Vertice, TipoGrafo *Grafo)
{ TipoApontador Aux = 0;  short ListaVazia = TRUE;
  while (Aux < Grafo->NumVertices && ListaVazia)
  { if (Grafo->Mat[*Vertice][Aux] > 0)
    ListaVazia = FALSE;
    else Aux++;
  }
  return (ListaVazia == TRUE);
}

TipoApontador PrimeiroListaAdj(TipoValorVertice *Vertice,
                               TipoGrafo *Grafo)
{ TipoValorVertice Result;
  TipoApontador Aux = 0;  short ListaVazia = TRUE;
  while (Aux < Grafo->NumVertices && ListaVazia)
    { if (Grafo->Mat[*Vertice][Aux] > 0)
      { Result = Aux; ListaVazia = FALSE; }
      else Aux++;
    }
  if (Aux == Grafo->NumVertices)
    printf("Erro: Lista adjacencia vazia (PrimeiroListaAdj)\n");
  return Result;
}

void ProxAdj(TipoValorVertice *Vertice, TipoGrafo *Grafo,
             TipoValorVertice *Adj, TipoPeso *Peso,
             TipoApontador *Prox, short *FimListaAdj)
{ /* Retorna Adj apontado por Prox */
  *Adj = *Prox;  *Peso = Grafo->Mat[*Vertice][*Prox];  (*Prox)++;
  while (*Prox < Grafo->NumVertices &&
         Grafo->Mat[*Vertice][*Prox] == 0) (*Prox)++;
  if (*Prox == Grafo->NumVertices)
  *FimListaAdj = TRUE;
}
```


Continuação do Programa G.3

```
void RetiraAresta(TipoValorVertice *V1, TipoValorVertice *V2,
                  TipoPeso *Peso, TipoGrafo *Grafo)
{ if (Grafo->Mat[*V1][*V2] == 0)
  printf("Aresta nao existe\n");
  else { *Peso = Grafo->Mat[*V1][*V2]; Grafo->Mat[*V1][*V2] = 0; }
}

void LiberaGrafo(TipoGrafo *Grafo)
{ /* Nao faz nada no caso de matrizes de adjacencia */ }

void ImprimeGrafo(TipoGrafo *Grafo)
{ short i, j;
  printf("    ");
  for (i = 0; i <= Grafo->NumVertices - 1; i++) printf("%3d", i);
  printf("\n");
  for (i = 0; i <=  Grafo->NumVertices - 1; i++)
    { printf("%3d", i);
      for (j = 0; j <=Grafo->NumVertices - 1; j++)
        printf("%3d", Grafo->Mat[i][j]);
      printf("\n");
    }
}
```

Programa G.4 *Estrutura do grafo com listas encadeadas usando apontadores*

```
#define MAXNUMVERTICES  100
#define MAXNUMARESTAS  4500

typedef int TipoValorVertice;
typedef int TipoPeso;
typedef struct TipoItem {
  TipoValorVertice Vertice;
  TipoPeso Peso;
} TipoItem;
typedef struct TipoCelula *TipoApontador;
struct TipoCelula {
  TipoItem Item;
  TipoApontador Prox;
} TipoCelula;
typedef struct TipoLista {
  TipoApontador Primeiro, Ultimo;
} TipoLista;
typedef struct TipoGrafo {
  TipoLista Adj[MAXNUMVERTICES + 1];
  TipoValorVertice NumVertices;
  short NumArestas;
} TipoGrafo;
```