Continuação do Programa G.24

```
TipoApontador PrimeiroListaInc(TipoValorVertice * Vertice,
                               TipoGrafo * Grafo)
{ TipoApontador ArestaAtual = 0;
  short Continua = TRUE; TipoApontador Resultado = 0;
  while (ArestaAtual < Grafo->NumArestas && Continua == TRUE)
    { if (Grafo->Mat[*Vertice][ArestaAtual] > 0)
      { Resultado = ArestaAtual; Continua = FALSE; }
      else ArestaAtual = ArestaAtual + 1;
    }
  if (ArestaAtual == Grafo->NumArestas)
  printf("Erro: Lista incidencia vazia\n");
  return Resultado;
}

void ProxArestaInc (TipoValorVertice * Vertice,
                    TipoGrafo * Grafo,
                    TipoValorAresta * Inc,
                    TipoPesoAresta * Peso,
                    TipoApontador * Prox,
                    short * FimListaAdj)
{ *Inc = *Prox;
  *Peso = Grafo->Mat[*Vertice][*Prox];
  *Prox = *Prox + 1;
  while (*Prox < Grafo->NumArestas &&
         Grafo->Mat[*Vertice][*Prox] == 0) *Prox = *Prox + 1;
  *FimListaAdj = (*Prox == Grafo->NumArestas);
}
```

Programa G.25 *Estrutura do tipo hipergrafo com listas de adjacência usando arranjos*

```
#define MAXNUMVERTICES 100
#define MAXNUMARESTAS 4500
#define MAXR 5
#define MAXTAMPROX MAXR * MAXNUMARESTAS
#define INDEFINIDO -1
typedef int TipoValorVertice;
typedef int TipoValorAresta;
typedef int Tipor;
typedef int TipoMaxTamProx;
typedef int TipoPesoAresta;
typedef TipoValorVertice TipoArranjoVertices[MAXR + 1];
typedef struct TipoAresta {
   TipoArranjoVertices Vertices;
   TipoPesoAresta Peso;
} TipoAresta;
typedef TipoAresta TipoArranjoArestas[MAXNUMARESTAS + 1];
```


Continuação do Programa G.25

```
typedef struct TipoGrafo {
  TipoArranjoArestas Arestas;
  TipoValorVertice Prim[MAXNUMARESTAS + 1];
  TipoMaxTamProx Prox[MAXTAMPROX + 2];
  TipoMaxTamProx ProxDisponivel;
  TipoValorVertice NumVertices;
  TipoValorAresta NumArestas;
  Tipor r;
} TipoGrafo;
typedef int TipoApontador;
```

Programa G.26 *Operadores implementados como listas de incidência usando arranjos*

```
short ArestasIguais(TipoArranjoVertices V1,
                    TipoValorAresta *NumAresta, TipoGrafo *Grafo)
{ Tipor i = 0, j;
  short Aux = TRUE;
  while (i < Grafo->r && Aux)
  { j = 0;
    while ((V1[i] != Grafo->Arestas[*NumAresta].Vertices[j]) &&
           (j < Grafo->r)) j++;
    if (j == Grafo->r) Aux = FALSE;
    i++;
  }
  return Aux;
}
void FGVazio(TipoGrafo *Grafo)
{ int i;
  Grafo->ProxDisponivel = 0;
  for (i = 0; i < Grafo->NumVertices; i++) Grafo->Prim[i] = -1;
}
void InsereAresta(TipoAresta *Aresta, TipoGrafo *Grafo)
{ int i, Ind;
  if (Grafo->ProxDisponivel == MAXNUMARESTAS + 1)
  printf ("Nao ha espaco disponivel para a aresta\n");
  else
  { Grafo->Arestas[Grafo->ProxDisponivel] = *Aresta;
    for (i = 0; i < Grafo->r; i++)
    { Ind = Grafo->ProxDisponivel + i * Grafo->NumArestas;
      Grafo->Prox[Ind] =
        Grafo->Prim[Grafo->Arestas[Grafo->ProxDisponivel].Vertices[i]];
      Grafo->Prim[Grafo->Arestas[Grafo->ProxDisponivel].Vertices[i]]=Ind;
    }
  }
  Grafo->ProxDisponivel++;
}
```