Programa G.6 *Estrutura do grafo com listas de adjacência usando arranjos*

```
#define MAXNUMVERTICES 100
#define MAXNUMARESTAS 4500
#define TRUE 1
#define FALSE 0
#define MAXTAM (MAXNUMVERTICES + MAXNUMARESTAS * 2)

typedef int TipoValorVertice;
typedef int TipoPeso;
typedef int  TipoTam;
typedef struct TipoGrafo {
  TipoTam Cab[MAXTAM + 1];
  TipoTam Prox[MAXTAM + 1];
  TipoTam Peso[MAXTAM + 1];
  TipoTam ProxDisponivel;
  char NumVertices;
  short NumArestas;
} TipoGrafo;
typedef short TipoApontador;
```

Programa G.7 *Operadores implementados como lista de adjacência usando arranjos*

```
void FGVazio(TipoGrafo *Grafo)
{ short i;
  for (i = 0; i <= Grafo->NumVertices; i++)
    { Grafo->Prox[i] = 0;  Grafo->Cab[i] = i;
      Grafo->ProxDisponivel = Grafo->NumVertices;
    }
}

void InsereAresta(TipoValorVertice *V1, TipoValorVertice *V2,
                  TipoPeso *Peso, TipoGrafo *Grafo)
{ short Pos;
  Pos = Grafo->ProxDisponivel;
  if (Grafo->ProxDisponivel == MAXTAM)
  { printf("nao ha espaco disponivel para a aresta\n"); return;
  }
  Grafo->ProxDisponivel++;
  Grafo->Prox[Grafo->Cab[*V1]] = Pos;
  Grafo->Cab[Pos] = *V2;  Grafo->Cab[*V1] = Pos;
  Grafo->Prox[Pos] = 0;  Grafo->Peso[Pos] = *Peso;
}

short ExisteAresta(TipoValorVertice Vertice1,
                   TipoValorVertice Vertice2, TipoGrafo *Grafo)
{ TipoApontador Aux;
  short EncontrouAresta = FALSE;
  Aux = Grafo->Prox[Vertice1];
```


Continuação do Programa G.7

```
  while (Aux != 0 && EncontrouAresta == FALSE)
    { if (Vertice2 == Grafo->Cab[Aux])
      EncontrouAresta = TRUE;
      Aux = Grafo->Prox[Aux];
    }
  return EncontrouAresta;
}

/* Operadores para obter a lista de adjacentes */
short ListaAdjVazia(TipoValorVertice *Vertice, TipoGrafo *Grafo)
{ return (Grafo->Prox[*Vertice] == 0); }

TipoApontador PrimeiroListaAdj(TipoValorVertice *Vertice,
                               TipoGrafo *Grafo)
{ return (Grafo->Prox[*Vertice]); }

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

void RetiraAresta(TipoValorVertice *V1, TipoValorVertice *V2,
                  TipoPeso *Peso, TipoGrafo *Grafo)
{ TipoApontador Aux, AuxAnterior;  short EncontrouAresta = FALSE;
  AuxAnterior = *V1;  Aux = Grafo->Prox[*V1];
  while (Aux != 0 && EncontrouAresta == FALSE)
    { if (*V2 == Grafo->Cab[Aux]) EncontrouAresta = TRUE;
      else { AuxAnterior = Aux; Aux = Grafo->Prox[Aux]; }
    }
  if (EncontrouAresta) /* Apenas marca como retirado */
  { Grafo->Cab[Aux] = MAXNUMVERTICES + MAXNUMARESTAS * 2;
  }
  else printf("Aresta nao existe\n");
}
void LiberaGrafo(TipoGrafo *Grafo)
{ /* Nao faz nada no caso de posicoes contiguas */ }

void ImprimeGrafo(TipoGrafo *Grafo)
{ short i, forlim;
  printf("     Cab Prox Peso\n");
  forlim = Grafo->NumVertices + Grafo->NumArestas * 2;
  for (i = 0; i <= forlim - 1; i++)
    printf("%2d%4d%4d%4d\n", i, Grafo->Cab[i],
           Grafo->Prox[i], Grafo->Peso[i]);
}
```