Implementação do TAD DiGrafo Ponderado usando Matriz de Adjacência

Profa. Dra. Denise Guliato
FACOM - UFU

Tipo do dado

```
typedef int TipoPeso;
typedef int TipoVertice;

struct grafo {
                    int NumVertices;
                    int NumArestas;
                    TipoPeso **Mat;
             };

typedef  struct grafo TipoGrafo;
```

TipoGrafo* Cria_grafo(int NVertices)

```
TipoGrafo* Cria_grafo(int NVertices){
  int i, k;
  TipoGrafo *Grafo ;
  if (Nvertices <= 0) return NULL;
  Grafo = (TipoGrafo*) malloc(sizeof(TipoGrafo));
  if (Grafo == NULL)  return NULL;
  Grafo->Mat = (TipoPeso **) malloc(NVertices*sizeof(TipoPeso*));
  if (Grafo->Mat == NULL)
  {    free(Grafo);
       return NULL;  }
  for(i=0; i<NVertices; i++)
  {
      Grafo->Mat[i] = (TipoPeso*) calloc(NVertices,sizeof(TipoPeso));
                     //aloca e preenche com zeros
      if (Grafo->Mat[i]  == NULL) {  // não conseguiu alocar a linha i da matriz
          for (k=0; k<i; k++)
             free(Grafo->Mat[k]);    //libera todas a linhas alocadas
          free(Grafo->Mat);
          free(Grafo) ;
          return NULL;
      }
  }
  Grafo->NumVertices = NVertices;
  Grafo->NumArestas = 0;
  return Grafo;
}
```

int Insere_Aresta(TipoGrafo *Grafo, TipoVertice v1, TipoVertice v2, TipoPeso peso)

```
int Insere_Aresta(TipoGrafo *Grafo, TipoVertice v1,
TipoVertice v2, TipoPeso peso){

   if (Grafo == NULL)
      Return -1;          // o grafo não existe
   if (v1<0 || v1>= Grafo->NumVertices || v2 < 0 ||
       v2 >= Grafo->NumVertices)
      return -1;          // não eh possivel criar aresta
   if (Grafo->Mat[v1][v2] != 0 || peso <= 0)
      return 0;           // já existe aresta entre v1 e v2
                          // ou peso invalido

   Grafo->Mat[v1][v2] = peso;
   Grafo->NumArestas ++;
   return 1;
}
```

int Existe_Aresta(TipoGrafo *Grafo, TipoVertice v1, TipoVertice v2)

```
int Existe_Aresta(TipoGrafo *Grafo, TipoVertice v1,
TipoVertice v2)
{
    if (Grafo == NULL)
       return -1;
    if (v1<0 || v1>= Grafo-> NumVertices || v2 < 0 ||
        v2 >= Grafo->NumVertices)
       return -1; // nao eh possivel criar aresta

    if( Grafo->Mat[v1][v2] == 0)
         return 0; //aresta nao existe ligando v1 e v2
    else return 1; //existe aresta ligando v1 e v2
}
```

int Retira_Aresta(TipoGrafo *Grafo, TipoVertice v1, TipoVertice v2)

```
int Retira_Aresta(TipoGrafo *Grafo, TipoVertice
v1, TipoVertice v2)
{
     if (Grafo == NULL)
       Return -1;              // grafo nao existe
     if (v1<0 || v1>=Grafo-> NumVertices ||
         v2<0 || v2>=Grafo-> NumVertices)
      return -1;    // nao eh possivel criar aresta

     if( Grafo->Mat[v1][v2] == 0)
         return 0;             // aresta nao existe

    Grafo->Mat[v1][v2] = 0; //remove aresta
    Grafo->NumArestas--;
    return 1;
}
```

int Consulta_Aresta(TipoGrafo *Grafo, TipoVertice v1, TipoVertice v2, TipoPeso* peso)

```
int Consulta_Aresta(TipoGrafo *Grafo, TipoVertice v1,
TipoVertice v2, TipoPeso* peso)
{
     if (Grafo == NULL)
         return -1;
     if (v1<0 || v1>= Grafo-> NumVertices ||
         v2<0 || v2>= Grafo-> NumVertices)
         return -1; // vertices invalidos

     if (Grafo->Mat[v1][v2] == 0)
         return 0;

     *peso = Grafo->Mat[v1][v2];
     return 1;
}
```

void Mostra_Lista_Adjacentes (TipoGrafo *Grafo, TipoVertice v)

```
void Mostra_Lista_Adjacentes(TipoGrafo *Grafo,TipoVertice v)
{
   int i, cont =0 ;
   if (Grafo == NULL || v<0 || v >= Grafo->NumVertices)
       printf("\n grafo nao existe ou vertice invalido");
   else  {
       printf("\n lista de vertices adjacentes a %4d:",v);
       for (i=0; i<Grafo->NumVertices; i++)
            if (Grafo->Mat[v][i] != 0)
            {
               printf("%4d -%4d", i, Grafo->Mat[v][i]);
               cont = 1;
            }
       if (cont == 0)
           printf("\nVertice %d nao possui adjacentes", v);
   }
}
```

void Mostra_Grafo(TipoGrafo *Grafo)

```
void Mostra_Grafo(TipoGrafo *Grafo)
{
   int i, j;
   if (Grafo == NULL || Grafo->NumArestas == 0)
       printf("\n grafo nao existe ou nao possui arestas");
   else
      for (i=0; i<Grafo->NumVertices; i++)
      {
        printf("\n vertices adjacente a %4d ----", i);
        for (j=0; j<Grafo->NumVertices; j++)
            if (Grafo->Mat[i][j] != 0)
                printf(" %4d, %d ;",j, Grafo->Mat[i][j]);
      }
}
```

TipoGrafo* Libera_Grafo(TipoGrafo* Grafo)

```
TipoGrafo* Libera_Grafo(TipoGrafo* Grafo)
{
   int i;
   if (Grafo == NULL)
      return NULL;

   for(i=0; i< Grafo->NumVertices; i++)
      free(Grafo->Mat[i]);

   free(Grafo->Mat);
   free(Grafo);
   Grafo = NULL;
   return Grafo;
}
```

Análise

- Deve ser utilizada para grafos **densos** em que $|A|$ é próximo de $|V|^2$;
- Rapidez para saber se existe uma aresta ligando dois vértices;
- Desvantagens:
  - a matriz necessita de $\Omega(|V|^2)$ de espaço;
  - a inspeção de toda a matriz tem complexidade $0(|V|^2)$