```
void remv (listaDeNodos node, int *listaDeNodosVazios, int p,
int q) {
   int r , r2;
   r2 = -1 ;
   r = node[p].point;
   while (r >= 0 && node[r].point != q) {
      r2 = r;
      r = node[r].next;
   }
   if (r >= 0) {
      /* r aponta para um arco de node[p] para node[q] */
      if (r2 < 0)
         node[p].point = node[r].next;
      else
         node[r2].next = node[r].next;
      freenode(listaDeNodosVazios, node, r);
      return;
   } /* na inexistência de um arco, nenhuma ação precisa ser
tomada */
}
```

Grafos - Representação

Implemente a operação **freenode()**.

```
void freenode (int *listaDeNodosVazios,
listaDeNodos node, int r)
{
  node[r].next = *listaDeNodosVazios;
  *listaDeNodosVazios = r;
}
```

Grafos - Representação

Implemente a operação **remvwt (node, &listaDeNodosVazios, p, q, x)** que define **x** com o peso do arco **<p,q>** num grafo ponderado e depois remove o arco do grafo.

```
void remvwt (listaDeNodos node, int *listaDeNodosVazios, int p, int
q, int x) {
   int r , r2;
   r2 = -1 ;
   r = node[p].point;
   while (r >= 0 && node[r].point != q) {
      r2 = r;
      r = node[r].next;
   }
   if (r >= 0) {
      /* r aponta para um arco de node[p] para node[q] */
      if (r2 < 0)
         node[p].point = node[r].next;
      else
         node[r2].next = node[r].next;
      node[r].info = x;
      freenode(listaDeNodosVazios, node, r);
   }
   /*na inexistência de um arco, nenhuma ação precisa ser tomada*/
}
```

Grafos - Representação

Implemente a operação **adjacent (node, p, q)** a qual aceita ponteiros para dois nós de cabeçalho e determina se **node(q)** é adjacente a **node(p)**.

```
char adjacent (listaDeNodos node, int p, int q)
{
   int r;
   r = node[p].point;
   while (r >= 0)
      if (node[r].point == q)
         return(1);
      else
         r = node[r].next ;
   return (0);
}
```

Grafos - Representação

Implemente a operação **findnode(node, graph, x)** que retorna um ponteiro para um nó de cabeçalho com o campo de informação **x**, caso exista esse nó de cabeçalho, e retorna o ponteiro nulo (com valor -1), caso contrário.

```
int findnode (listaDeNodos node, int graph, int x)
{
   int p;
   p = graph;
   while (p >= 0)
      if (node[p].info == x)
         return(p);
      else
         p = node[p].next;
      return (-1);
}
```

Grafos - Representação

Implemente a operação **addnode (node, &listaDeNodosVazios, &pgraph, x)** inclui um nó com o campo de informação **x** num grafo e retorna um ponteiro para esse nó.

UNIVASF

```
int addnode (listaDeNodos node,
int *listaDeNodosVazios, int* pgraph, int x) {
   int p;
   p = getnode(listaDeNodosVazios, node);
   node[p].info = x;
   node[p].point = -1;
   node[p].next = *pgraph;
   *pgraph = p;
   return (p);
}
```

Grafos - Representação

É importante salientar outra diferença relevante entre a representação de matriz de adjacência e a representação ligada de grafos.

Na representação de matriz está implícita a possibilidade de percorrer uma linha ou coluna da matriz. Percorrer uma linha equivale a identificar todos os arcos emanando de determinado nó. Isso pode ser feito com eficiência na representação ligada, percorrendo a lista de nós de arco, a partir de determinado nó de cabeçalho.

Entretanto, percorrer uma coluna de uma matriz de adjacência é equivalente a identificar todos os arcos que terminam em determinado nó;

UNIVASF

Grafos - Representação

não existe um método correspondente para fazer isso sob a representação ligada que trabalhamos.