Grafos - Representação

A representação ligada poderia ser alterada de modo a incluir duas listas emanando de cada nó de cabeçalho: uma para os arcos emanando do nó de grafo e outra para os arcos terminando no nó de grafo.

Entretanto, isso exigiria a alocação de dois nós para cada arco, aumentando, por conseguinte, a complexidade da inclusão ou eliminação de um arco.

Como alternativa, cada nó de arco poderia ser colocado em duas listas.

UNIVASF

Grafos - Representação

Nesse caso, um nó de arco conteria quatro ponteiros: um para o próximo arco emanando do mesmo nó, um para o próximo arco terminando no mesmo nó, um para o nó de cabeçalho no qual ele termina e um para o nó de cabeçalho a partir do qual ele emana. Um nó de cabeçalho conteria três ponteiros: um para o próximo nó de cabeçalho, um para a lista de arcos emanando a partir dele e outro para a lista de arcos terminando nesse nó.

Evidentemente, o programador precisará escolher entre essas representações examinando as necessidades do problema específico e considerando a eficiência em termos de tempo e espaço de armazenamento.

UNIVASF

Grafos - Representação

Implemente uma rotina **remvnode(..., &graph, p)** que remova um nó de cabeçalho apontado por **p** de um grafo apontado por **graph**, usando a representação ligada em vetor.

Evidentemente, quando um nó é removido de um grafo, todos os arcos emanando e terminando nesse nó precisarão ser removidos também.

Na representação ligada que estamos utilizando, não existe um método fácil de remover um nó de um grafo porque os arcos terminando no nó não podem ser obtidos diretamente.

UNIVASF

```
int remvnode(int *listaDeNodosVazios, listaDeNodos node, int *graph, int
p) {
  int i;
  int nodoAtual, nodoAnterior, retorno = 0;
  nodoAtual = nodoAnterior = *graph;
  while (nodoAtual >= 0) {
    if (nodoAtual == p)
    {/*remover todas as arestas com origem no vértice e o próprio vértice*/
      int nodoAux, nodoAux2;
      if (nodoAtual == *graph) /*eliminado o vértice do conjunto de vértices*/
        *graph = node[*graph].next;
      else
        node[nodoAnterior].next = node[nodoAtual].next;
      nodoAux = node[nodoAtual].point;
      while (nodoAux >= 0) {
        nodoAux2 = nodoAux;
        nodoAux = node[nodoAux].next;
        freenode(listaDeNodosVazios, node, nodoAux2);/*eliminado arestas*/
      }
      nodoAnterior = nodoAtual;
      nodoAtual = node[nodoAtual].next;
      freenode(listaDeNodosVazios, node, p); /*eliminando vértice*/
      retorno = 1;
  }
```

```
    else
    {/*remover as arestas que levam ao vértice sendo removido, caso
existam*/
       int auxAnterior, auxAtual, nodeAux;
       auxAnterior = auxAtual = node[nodoAtual].point;
       while (auxAtual >=0) {
          if (node[auxAtual].point == p) {
             if (auxAtual == auxAnterior)
                node[nodoAtual].point = node[auxAtual].next;
             else
                node[auxAnterior].next = node[auxAtual].next;
             nodeAux = auxAtual;
             auxAnterior = auxAtual;
             auxAtual = node[auxAtual].next;
             freenode(listaDeNodosVazios, node, nodeAux);
          }
          else {
              auxAnterior = auxAtual;
              auxAtual = node[auxAtual].next;
          }
       }
       nodoAnterior = nodoAtual;
       nodoAtual = node[nodoAtual].next;
    }
  }
  return (retorno);
}
```

Grafos - Representação

Em resumo implementamos o TAD grafo com representação encadeada com a estrutura e operações a seguir:

```
#define MAXNODES 500
typedef struct nodetype {
   int info;
   int point ;
   int next;
}tipoNodo;
typedef tipoNodo listaDeNodos[MAXNODES];
void inicializarGrafo (int *graph);
void criaListaDeNodosVazios (int *listaDeNodosVazios,
listaDeNodos node);
int getnode (int *listaDeNodosVazios, listaDeNodos node);
void freenode (int *listaDeNodosVazios, listaDeNodos node,
int r);
```

Grafos - Representação

```
void joinwt (listaDeNodos node, int *listaDeNodosVazios,
int p, int q, int wt);
void join (listaDeNodos node, int *listaDeNodosVazios, int p,
int q);
void remv (listaDeNodos node, int *listaDeNodosVazios, int p,
int q);
void remvwt (listaDeNodos node, int *listaDeNodosVazios,
int p, int q, int x);
char adjacent (listaDeNodos node, int p, int q);
int findnode (listaDeNodos node, int graph, int x);
int addnode (listaDeNodos node, int *listaDeNodosVazios,
int* pgraph, int x);
int remvnode(int *listaDeNodosVazios, listaDeNodos node,
int *graph, int p);
```

Grafos - Representação

Agora iremos implementar o TAD grafo com representação encadeada com a estrutura e operações a seguir:

```
typedef struct _nodetype {
   int info;
   struct _nodetype *point;
   struct _nodetype *next;
} nodetype;
typedef nodetype *TADgraph;
```