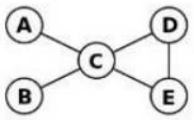
Universidade Federal de Uberlândia
Algoritmos e Programação - GGI

Roteiro para Laboratório - GRAFOS

PARTE 1 - Grafos - Implementação de uma matriz de adjacências (simples) para um grafo não direcionado.

Exemplo de Grafo não direcionado:



	A	B	C	D	E
A	0	0	1	0	0
B	0	0	1	0	0
C	1	1	0	1	1
D	0	0	1	0	1
E	0	0	1	1	0

Um grafo não direcionado é um grafo **simétrico**, pois para cada arco (U,V) existe um grafo (V,U). A matriz acima em linguagem C pode ser definida por:

```
int mat[5][5] = { {0,0,1,0,0},
                  {0,0,1,0,0},
                  {1,1,0,1,1},
                  {0,0,1,0,1},
                  {0,0,1,1,0} };
```

Assim, pode-se definir o algoritmo abaixo para o preenchimento de um grafo não direcionado, a partir da criação de uma matriz N x N:

```
// Matriz de Adjacências para um GRAFO NÃO DIRECIONADO
// Sendo grafo não orientado, o arco 1-2 significa 2-1 também.

// Preenchimento inicial do Grafo com zeros
para i <- 0 até N-1, faça
      para j <- 0 até N-1, faça
            matriz[i][j] <- 0
      fim-para
fim-para

// Leitura dos arcos
enquanto (recebe x, y e x <> -1), faça
      matriz[x][y] <- 1
      matriz[y][x] <- 1
fim-enquanto
```

Depois de construída a **matriz de adjacências**, representando os **arcos** de um grafo (aqui não direcionado), é possível responder à uma série de perguntas sobre o grafo, como por exemplo, o **grau** de cada vértice. No exemplo dado neste roteiro, o grau do vértice C é 4.

Codificação Inicial em C:

```c
#include <stdio.h>
#define NMAX 101


int main() {
	int n, i, j, x, y;
	int g[NMAX][NMAX];   // matriz estática 101 x 101

	printf("Informe o tamanho da matriz (N x N): ");
	scanf("%d",&n);
	printf("OK. Tamanho da matriz: %d \n",n);

	for (i=0; i<n; i++) {
		for (j=0; j<n; j++) {
			g[i][j]=0;
		}
	}

	printf("\nMatriz Inicial: \n");
	for (i=0; i<n; i++) {
		for (j=0; j<n; j++) {
			printf("[%d][%d]= %d \t",i,j,g[i][j]);
		}
		printf("\n");
	}

	printf("\nInforme os arcos: (-1) para encerrar\n");
	printf("\nVertice de partida: ");
	scanf("%d",&x);
	while (x != -1){
		printf("\nVertice de chegada: ");
		scanf("%d",&y);
		g[x][y] = 1;
		g[y][x] = 1;
		printf("\nVertice de partida: ");
		scanf("%d",&x);
	}

	printf("\nMatriz Final: \n");
	for (i=0; i<n; i++) {
		for (j=0; j<n; j++) {
			printf("[%d][%d]= %d \t",i,j,g[i][j]);
		}
		printf("\n");
	}

	system("pause");
	return 0;
}
```

Exercícios:

1) Reescreva o código fazendo a seguinte modularização:
   (a) Função para construção do grafo inicial, com apenas zeros.
   (b) Função para o preenchimento dos arcos existentes.

2) Faça a função para mostrar o grau de um vértice.

```
// Algoritmo para calcular e imprimir o grau de todos os vértices
para i <- 0 até N-1, faça
      grau <- 0
      para j <- 0 até N-1, faça
            se matriz[i][j] = 1
                  então grau <- grau + 1
            fim-se
      fim-para
      imprima "O vertice " i "tem grau" grau "."
fim-para
```

Obs: perceba que a utilização de matriz de adjacências gera a necessidade de um laço dentro de outro laço. Isso gera um custo O(n^2).
O custo O(n^2) é grande. E se for uma matriz com milhões de vértices? (Amigos no Facebook?)

3) Faça uma função para eliminar um arco específico em um grafo (baseado na matriz de adjacências).

4) Refaça o programa para que a matriz de adjacências implementada seja correspondente à um grafo direcionado.

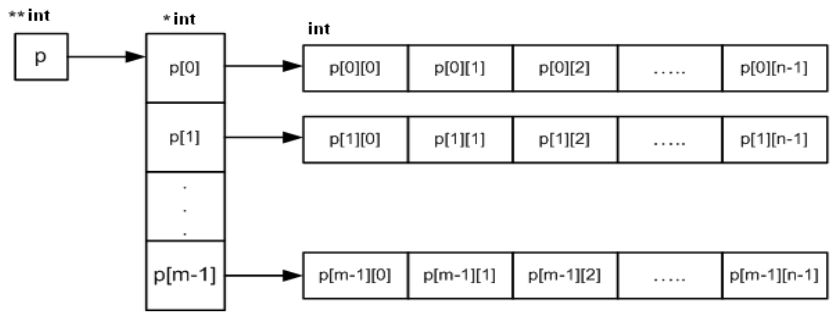
PARTE 2 - Grafos - Implementação de uma matriz de adjacências para um grafo não direcionado usando estruturas dinâmicas (alocação dinâmica de memória).

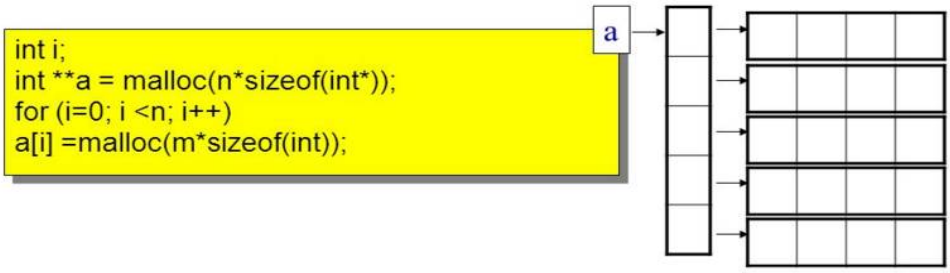
Definição de Grafo:

```
struct grafo {
            int NumVertices;
            int NumArestas;
            int **p;
       };

typedef struct grafo TipoGrafo;
```

Estrutura dinâmica a ser construída a partir do número de vértices para o campo "p":

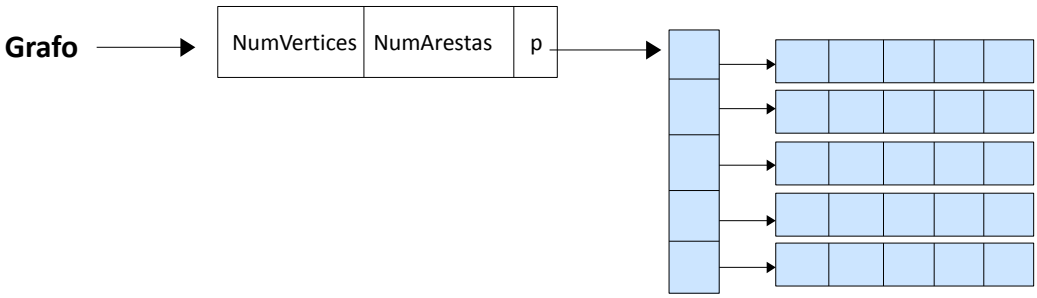




n = quantidade de linhas
m = quantidade de colunas

Função de alocação de memória para o Grafo:

- A função devolve a posição de memória de início para esta alocação.
- A função recebe a quantidade de vértices do grafo (N) para que possa ser feita a alocação de memória para a matriz (N x N)




```
TipoGrafo* Aloca_grafo(int NVertices)
{
  int i, k;
  TipoGrafo *Grafo ;

  Grafo = (TipoGrafo*) malloc(sizeof(TipoGrafo));
  // Aloca estrutura inicial (numVertices, numArestas, p)

  Grafo->p = (int **) malloc(NVertices*sizeof(int*));
  // Aloca um vetor de ponteiros para cada linha da matriz

  for(i=0; i<NVertices; i++)
  {
     Grafo->p[i] = (int*) calloc(NVertices,sizeof(int));
     // Aloca e preenche com zeros cada linha da matriz
  }
  Grafo->NumVertices = NVertices;
  Grafo->NumArestas = 0;
  return Grafo;
}
```

Exercícios:

1) Monte um projeto usando DevC++ para manter as funções de manipulação de um grafo não direcionado utilizando matriz de adjacências e alocação dinâmica, de acordo com o tipo de dado (TAD) visto.

2) Faça uma função para percorrer a matriz de adjacências.

3) Faça uma função para armazenar os arcos na matriz de adjacências.