DISCIPLINA: ESTRUTURA DE DADOS II

2018

UEA
UNIVERSIDADE DO ESTADO DO AMAZONAS

Implementação de Grafos

Prof. Luis Cuevas Rodríguez, PhD

EST
Escola Superior de Tecnologia da UEA

NUCOMP
NÚCLEO DE COMPUTAÇÃO

Grafos

- Analises de código (Lista de Adjacência)
- Representação de Grafos usando Matriz de Adjacência
- Criação de um grafo
- Incluir uma aresta no grafo direcionado
- Excluir uma aresta no grafo direcionado
- Visualizar o grafo

Analises de código

1. O código para representar um grafo usando listas de adjacência mostrado na sala de aula é para uma arvore direcionada. Modifique o código para representar um grafo não direcionado.
2. Programar funções para determinar

a. Dado um no, qual é o grau desses no?

b. ***Calcula o caminho entre dois no.***

c. Dados dois no, procurar se existe uma aresta entre eles.

d. Dado um no mostrar seu grau de entrada

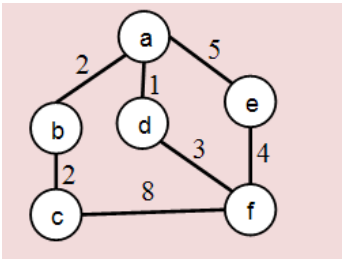
Representar um grafo não direcionado

```
bool criaAresta_nao(GRAFO* gr, int vi, int vf, TIPOPESO p){
 if (!gr) return (false);
 if ((vf<0)|| (vf >= gr->vertices))
     return (false);
 if ((vi<0)|| (vi>=gr->vertices))
     return (false);
 ADJACENCIA* novo = criaAdj(vf,p);
 novo->prox=gr->vert[vi].cad;
 gr->vert[vi].cad = novo;

 ADJACENCIA* novo2 = criaAdj(vi,p);
 novo2->prox=gr->vert[vf].cad;
 gr->vert[vf].cad = novo2;

 gr->arestas++;
 return (true);
}
```

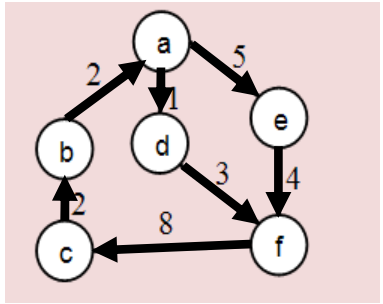
Dado um no, qual é o grau desses no

```
int grau_do_no(GRAFO* gr, int no){
  int i = no-1, c=0;
  ADJACENCIA* ad = gr->vert[i].cad;
  while(ad){
      c++;
      ad = ad->prox;
  }
  return c;
}
```

va ➔	vb,2 ➔ vd,1 ➔ ve,5
vb ➔	va,2 ➔ vc,2
vc ➔	vb,2 ➔ vf,8
vd ➔	va,1 ➔ vf,3
ve ➔	va,5 ➔ vf,4
vf ➔	ve,4 ➔ vd,3 ➔ vc,8



va ➔	vd,1 ➔ ve,5
vb ➔	va,2
vc ➔	vb,2
vd ➔	vf,3
ve ➔	vf,4
vf ➔	vc,8

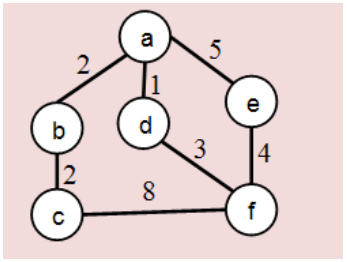


UEA
UNIVERSIDADE DO ESTADO DO AMAZONAS

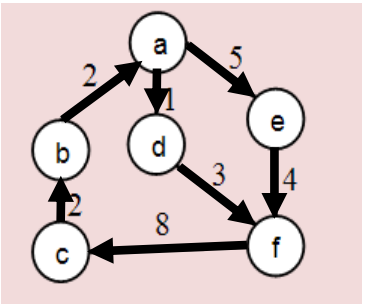
Dados dois no, procurar se existe uma aresta entre eles.

```
bool existe_aresta(GRAFO* gr, int vi, int vf){
      ADJACENCIA* ad = gr->vert[vi].cad;
      while(ad){
         if (ad->vertice == vf) return true;
         ad = ad->prox;
      }
     return false;
}
```

va →	vb,2 → vd,1 → ve,5
vb →	va,2 → vc,2
vc →	vb,2 → vf,8
vd →	va,1 → vf,3
ve →	va,5 → vf,4
vf →	ve,4 → vd,3 → vc,8



va →	vd,1 → ve,5
vb →	va,2
vc →	vb,2
vd →	vf,3
ve →	vf,4
vf →	vc,8

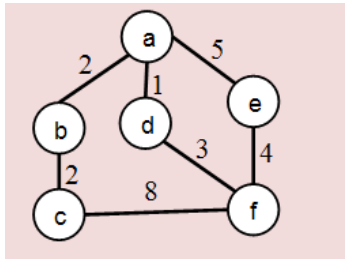


UEA
UNIVERSIDADE DO ESTADO DO AMAZONAS

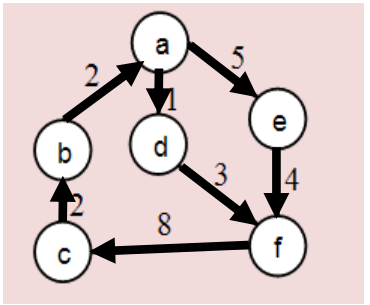
Dado um no mostrar seu grau de entrada

```
int grau_no_entrada(GRAFO* gr, int no){
  int j = no-1, c=0;
  for (int i=0; i < gr->vertices; i++){
     ADJACENCIA* ad = gr->vert[i].cad;
     while(ad){
        if (ad->vertice == j) c++;
        ad = ad->prox;
     }
   }
  return c;
}
```

va ➔	vb,2 ➔ vd,1 ➔ ve,5
vb ➔	va,2 ➔ vc,2
vc ➔	vb,2 ➔ vf,8
vd ➔	va,1 ➔ vf,3
ve ➔	va,5 ➔ vf,4
vf ➔	ve,4 ➔ vd,3 ➔ vc,8



va ➔	vd,1 ➔ ve,5
vb ➔	va,2
vc ➔	vb,2
vd ➔	vf,3
ve ➔	vf,4
vf ➔	vc,8



UEA
UNIVERSIDADE DO ESTADO DO AMAZONAS

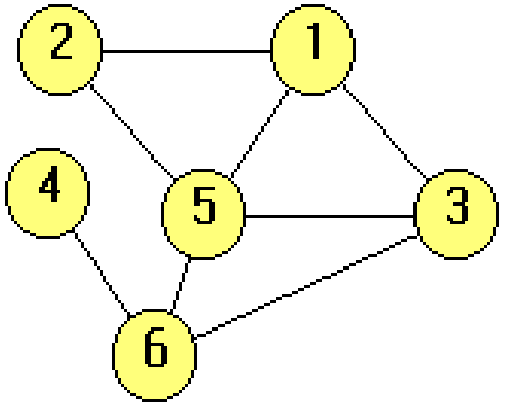
Grafos – Matriz de Adjacência

- Uma matriz *n x n (|V|x|V|)*, onde n é o numero de vértices.
- A[i,j] ➔ representa se houver uma aresta desde o vértice *i* até o vértice *j*

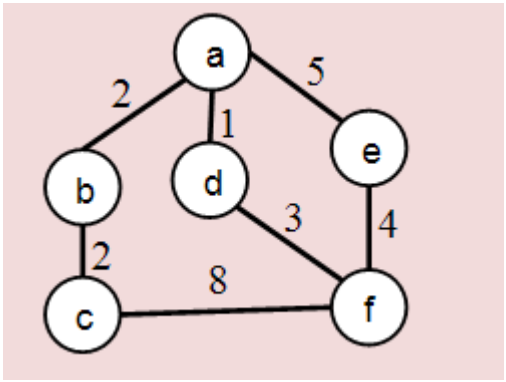
$$a_{ij} = \begin{cases} 1 & \text{se } (i,j) \in E, \\ 0 & \text{em caso contrário} \end{cases}$$

  - Se o grafo é ponderado o valor de a[i,j] é o peso
  - Grafo não ponderado o valor de a[i,j] é 0 ou 1

Grafos – Matriz de Adjacência



0	1	1	0	1	0
1	0	0	0	1	0
1	0	0	0	1	1
0	0	0	0	0	1
1	1	1	0	0	1
0	0	1	1	1	0



	a	**b**	**c**	**d**	**e**	**f**
a	0	2	0	1	5	0
b	2	0	2	0	0	0
c	0	2	0	0	0	8
d	1	0	0	0	0	3
e	5	0	0	0	0	4
f	0	0	8	3	4	0

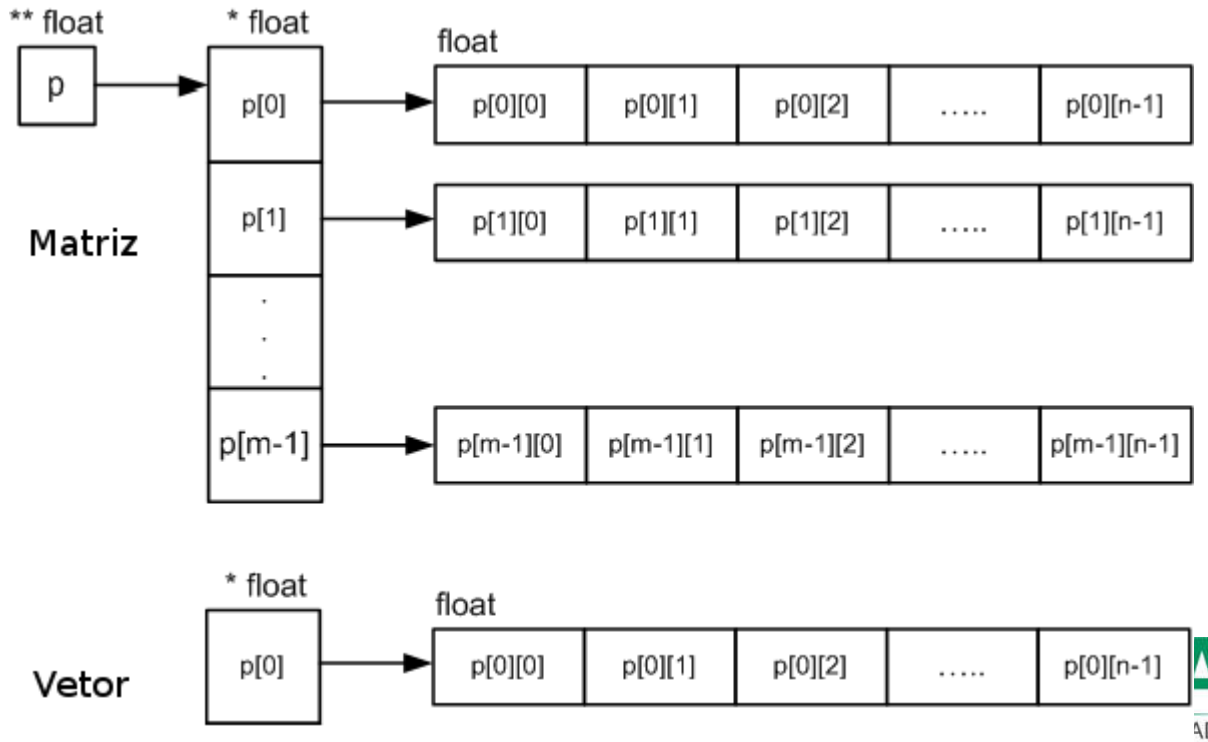
Prof. Luis Cuevas Rodríguez, PhD

UEA
UNIVERSIDADE DO ESTADO DO AMAZONAS

Implementação

- Tipo de Dado Abstrato

```
typedef struct grafo{
int vertices;
int arestas;
int **adj;
}GRAFO;
```



Criar e inicializar grafo

```
GRAFO* inicializa_grafo(int v) {
    GRAFO* g = new (GRAFO);
    g->vertices = v,
    g->arestas = 0;
    //incializar matrix
    g->adj = new (int* [v]);
    for (int i = 0; i < v; i++)
        g->adj[i] = new (int [v]);
    for (int i = 0; i < v; i++)
        for (int j = 0; j < v; j++)
            g->adj[i][j]=0;
    return g;
}
```

v=5

```
Vertice 0:  0 0 0 0 0
Vertice 1:  0 0 0 0 0
Vertice 2:  0 0 0 0 0
Vertice 3:  0 0 0 0 0
Vertice 4:  0 0 0 0 0
```

- v: numero de vértices

Inserir uma aresta

```
bool inserir_aresta(GRAFO* gr, int vi, int vf) {
     if (!gr) return (false);
     if ((vf<0)|| (vf >= gr->vertices))
          return (false);
     if ((vi<0)|| (vi>=gr->vertices))
          return (false);
     if (gr->adj[vi][vf] == 0) {
       gr->adj[vi][vf] = 1;
       gr->arestas++;
     }
     return true;
}
```

```
Vertice 0:  0 0 0 0 0
Vertice 1:  1 0 1 0 0
Vertice 2:  1 0 0 0 1
Vertice 3:  0 1 0 0 0
Vertice 4:  0 0 0 1 0
```

```
int main()
{
     GRAFO* gr=inicializa_grafo(5);
     imprime(gr);
     inserir_aresta(gr,1,0);
     inserir_aresta(gr,1,2);
     inserir_aresta(gr,2,0);
     inserir_aresta(gr,2,4);
     inserir_aresta(gr,3,1);
     inserir_aresta(gr,4,3);
     imprime(gr);
```

Excluir uma aresta

```
bool apagar_aresta(GRAFO* gr, int vi, int vf) {
    if (!gr) return (false);
    if ((vf<0)|| (vf >= gr->vertices))
        return (false);
    if ((vi<0)|| (vi>=gr->vertices))
        return (false);
    if (gr->adj[vi][vf] == 1) {
      gr->adj[vi][vf] = 0;
      gr->arestas--;
   }
   return true;
}
```

```
Vertice 0:  0 0 0 0 0
Vertice 1:  1 0 1 0 0
Vertice 2:  1 0 0 0 1
Vertice 3:  0 0 0 0 0
Vertice 4:  0 0 0 1 0
```

```
apagar_aresta(gr,3,1);
imprime(gr);
```

Imprimir Matriz

```
void imprime(GRAFO* gr){
    for (int i = 0; i < gr->vertices; i++) {
      cout << endl << "Vertice " << i << ": " ;
      for (int j = 0; j < gr->vertices; j++)
         cout << " " << gr->adj[i][j];
    }
}
```

```
Vertice 0:  0 0 0 0 0
Vertice 1:  1 0 1 0 0
Vertice 2:  1 0 0 0 1
Vertice 3:  0 0 0 0 0
Vertice 4:  0 0 0 1 0
```

Imprimir Arestas

```
void imprime_arestas(GRAFO* gr){

    for (int i = 0; i < gr->vertices; i++) {
      for (int j = 0; j < gr->vertices; j++){
         if (gr->adj[i][j] == 1)
            cout << "(" << i << ", " << j << ")";
      }
    }
}
```

```
Vertice 0:  0 0 0 0 0
Vertice 1:  1 0 1 0 0
Vertice 2:  1 0 0 0 1
Vertice 3:  0 0 0 0 0
Vertice 4:  0 0 0 1 0
Arestas
(1, 0)(1, 2)(2, 0)(2, 4)(4, 3)
```

Exercícios

1. Baseado no código anterior definir as funções de inserir e excluir para um grafo não direcionado.
2. Criar uma estrutura de dados baseada no código para grafo ponderados.
3. Programar funções para determinar:
   a. Dado um no, qual é o grau desses no?
   b. Dados dois no, procurar se existe uma aresta entre eles.
   c. Dado um no mostrar seu grau de entrada (direcionado)