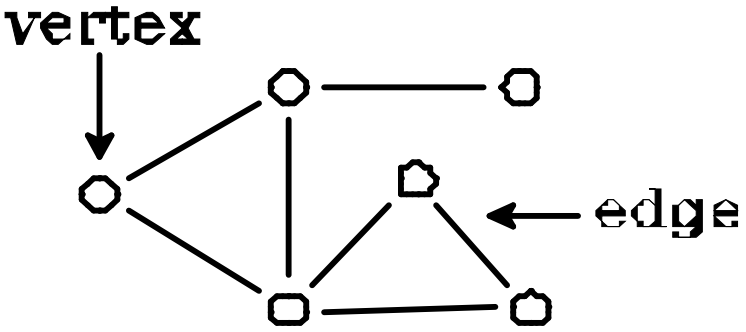
ICMC/USP
SCC0216 - Modelagem Computacional em Grafos

LAB 01 - TAD e operações comuns

Prof. Dr. Alneu de Andrade Lopes
1 sem. 2017
PAE: Alan Valejo

Introdução

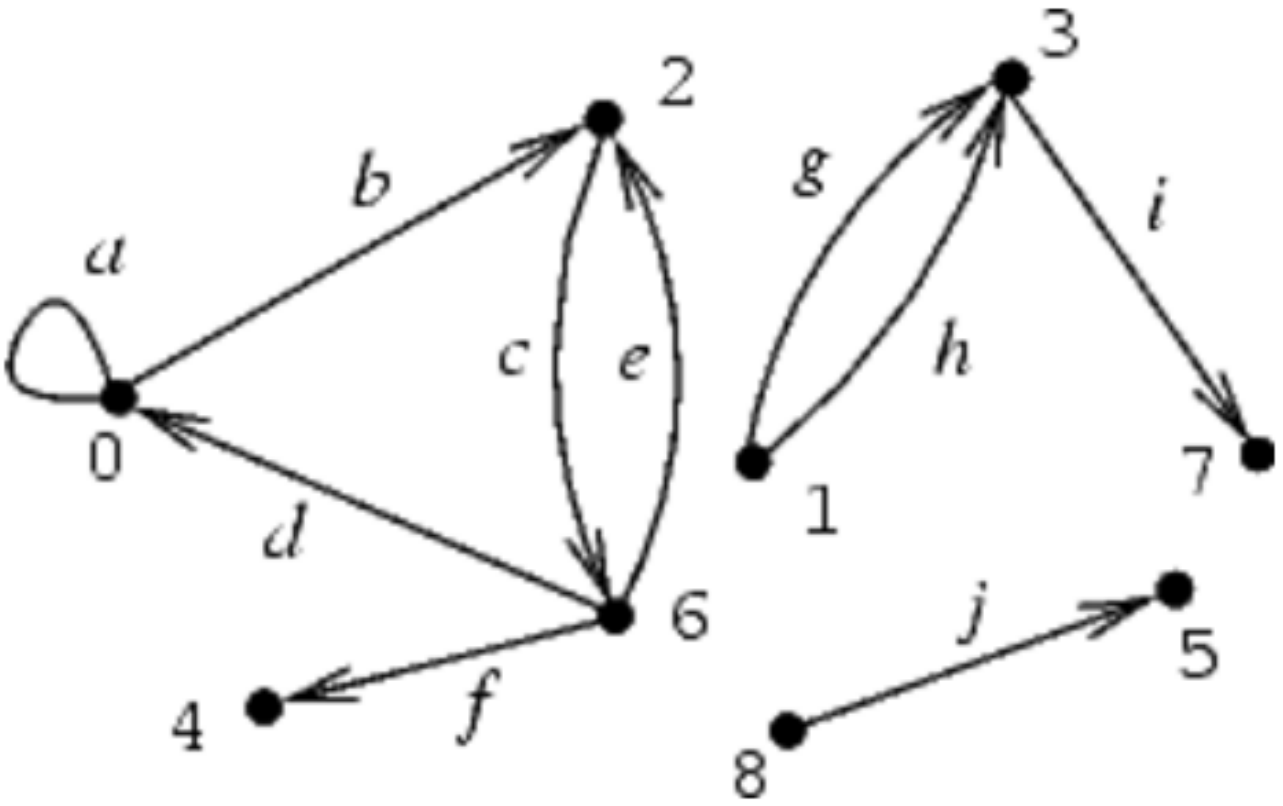
- Grafos → Ferramenta matemática utilizada para modelar problemas ou modelar sistemas e suas interações.



Grafo *G={V,E}*
- Coleção de vértices *V*
- Coleção de arestas *E*

Por exemplo:
- Rede Social
- Vértices: Pessoas
- Arestas: Relacionamento de amizade

Introdução



- Direcionado
- Laço ou auto-loop
- Desconexo

TAD Grafos

- Inicialmente
  - Introdução ferramenta matemática Grafo
  - Principais componentes dessa ferramenta
- Armazenamento e organização dos dados
  - Abstrair as operações primitivas úteis
  - Agrupa a estrutura de dados juntamente com as operações que podem ser feitas sobre esses dados
- Usuário só “enxerga” a interface, não a implementação
  - Importante na implementação de algoritmos
- Teoria dos Grafos: Estudo de caso
  - Requisito, estrutura de dados 1
  - Demostra a importância do TAD

Introdução

- No decorrer do curso
  - Teoria dos Grafos: O estudo das propriedades, algoritmos e aplicações.
- TAD
  - Auxilia na apresentação do conhecimento
  - A literatura parte do princípio que o leitor já abstraiu as operações primitivas
  - A escolha de uma estrutura de dados apropriada torna um problema complexo em uma solução simples
  - Permite a melhor compreensão dos algoritmos e maior facilidade de programação
  - Torna o código mais enxuto

Operações básicas

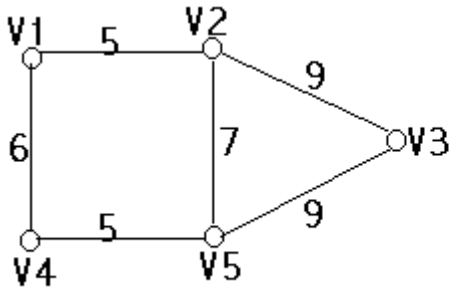
1. Criar um grafo vazio.
2. Inserir uma aresta no grafo.
3. Verificar se existe determinada aresta no grafo.
4. Obter a lista de vértices adjacentes a determinado vértice.
5. Retirar uma aresta do grafo.
6. Imprimir um grafo.
7. Obter o número de vértices do grafo.
8. Obter o transposto de um grafo direcionado.
9. Obter a aresta de menor peso de um grafo.

Operações básicas

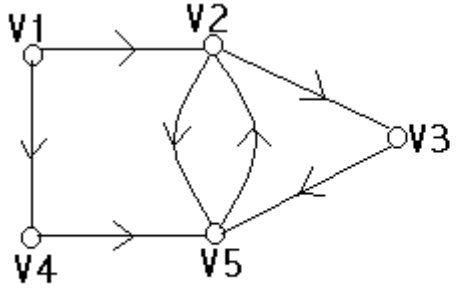
- Igraph
- http://igraph.org/python/doc/igraph.Graph-class.html

add_edge(source, target, **kwds) Adds a single edge to the graph.
add_edges(es) Adds some edges to the graph.
add_vertex(name=None, **kwds) Adds a single vertex to the graph.
add_vertices(n) Adds some vertices to the graph.
adjacent(vertex, mode=OUT) Returns the edges a given vertex is incident on.
as_directed(*args, **kwds) Returns a directed copy of this graph.
as_undirected(*args, **kwds) Returns an undirected copy of this graph.
delete_edges(self, *args, **kwds) Deletes some edges from the graph.
indegree(self, *args, **kwds) Returns the in-degrees in a list.
outdegree(self, *args, **kwds) Returns the out-degrees in a list.

Estrutura de Dados: Matriz de Adjacências



	V1	V2	V3	V4	V5
V1	0	5	0	6	0
V2	5	0	9	0	7
V3	0	9	0	0	9
V4	6	0	0	0	5
V5	0	7	9	5	0



	V1	V2	V3	V4	V5
V1	0	1	0	1	0
V2	0	0	1	0	1
V3	0	0	0	0	1
V4	0	0	0	0	1
V5	0	1	0	0	0

Estrutura de Dados: Matriz de Adjacências

Acesso a dados

n = número de vértices

M[i,j] = 1, se existir aresta de i a j

M[i,j] = 0, se NÃO existir aresta de i a j

Estrutura de Dados Simples

```
struct graph {
	int n;
	int **matrix;
};
```

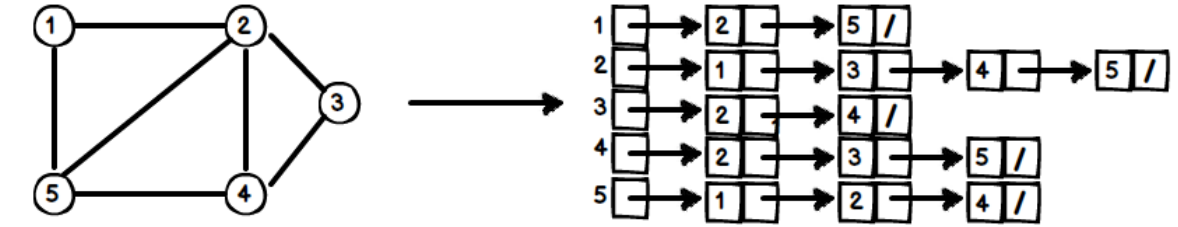
Estrutura de dados Elaborada

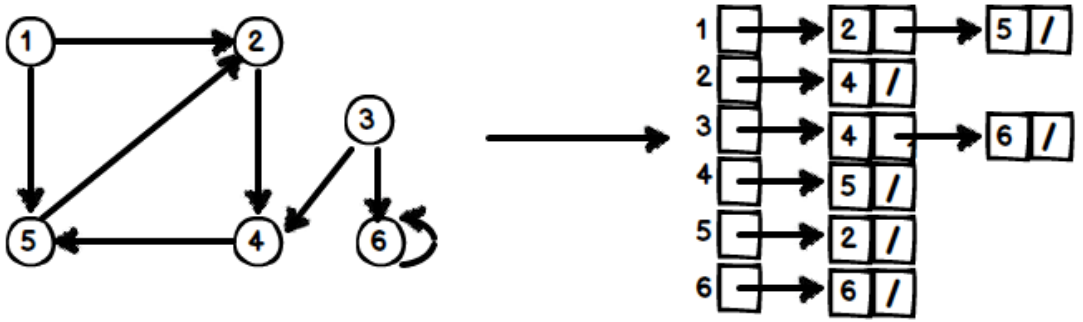
```
struct Node {
	int id;
	int degree;
	/* Informações */
};

struct Edge {
	int adj;
	int weight;
	/* Informações */
};

struct graph {
	int n;
	Node *nodes;
	Edge **edges;
};
```

Exemplo primeira função

```
Graph createGraph(int n) {
    Graph g = (Graph) malloc(sizeof(struct graph));
    g->n = n;
    g->matrix = (int **) malloc(n * sizeof(int *));
    int i;
        for(i = 0; i < n; i++) {
            g->matrix[i] = (int *) malloc(n * sizeof(int));
            int j;
            for(j = 0; j < n; j++) {
                g->matrix[i][j] = -1;
            }
        }
    return(g);
}
```

Estrutura de dados: Lista de adjacência





Estrutura de dados: Lista de adjacência

(1) Tradicional

```
struct graph {
        struct node *list;
        int n;
};

struct node {
        int info;
        struct node *next;
        struct arc *arcs;
};

struct arc {
        int weight;
        struct arc *next;
};
```

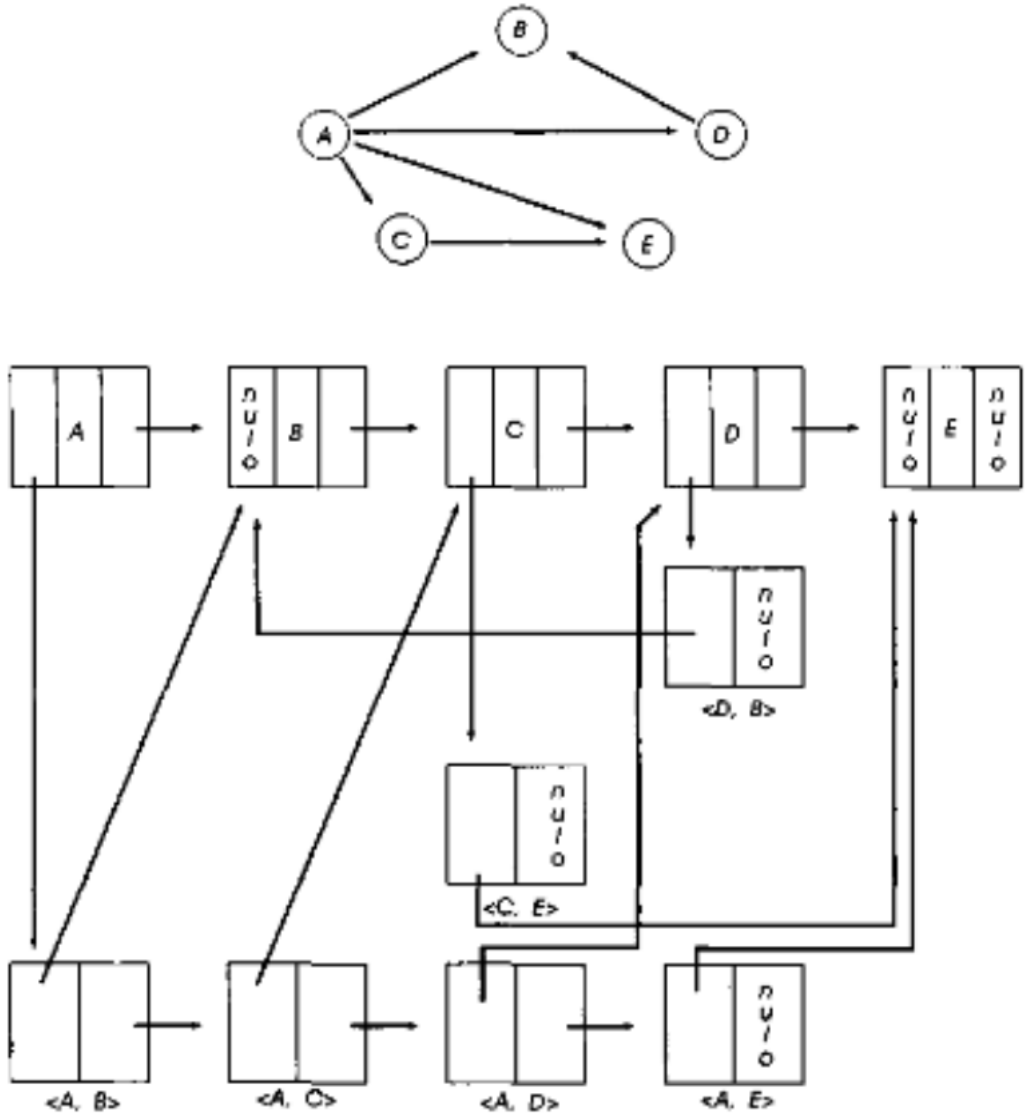
(2) Variação para grafos não ponderados

```
struct graph {
        struct node *list;
        int n;
};

struct node {
        int adj;
        struct node *next;
};
```

(3)

```
struct graph {
        struct node list[MAX];
        int n;
};

struct node {
        int adj;
        struct node *next;
        struct node *ant;
        struct node *first;
};
```

Estrutura de dados: Lista de adjacência



Estrutura de dados Complexa

```
struct graph {
        struct node *list;
        int n;
}

struct node {
        int info;
        struct node *next;
        struct node *arc;
};

struct arc {
        int weight;
        struct node *adj;
        struct arc *next;
};
```

Características

Dado as estruturas básicas

- Matriz de Adjacência
  - Armazenamento: O(n2)
  - Teste se aresta (i,j) está no grafo: O(1)
- Lista de Adjacência
  - Armazenamento: O(m + n)
  - Teste se aresta (i,j) está no grafo: O(d), com d sendo o grau do vértice i