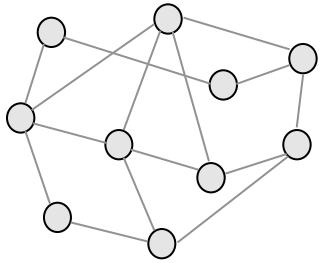
Universidade Estadual de Maringá
Centro de Tecnologia – Departamento de Informática



Introdução à Algoritmos em Grafos

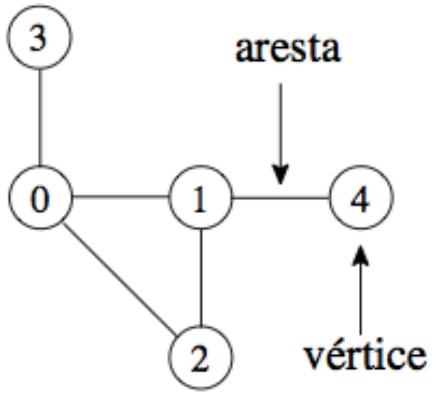
Prof. Heloise Manica Paris Teixeira

Introdução

- Muitas aplicações em computação necessitam considerar conjunto de conexões entre pares de objetos:
  - Web (documentos e suas conexões)
  - Existe um caminho para ir de um objeto a outro seguindo as conexões?
  - Qual é a menor distância entre um objeto e outro objeto?
  - Quantos outros objetos podem ser alcançados a partir de um determinado objeto?
- Existe um tipo abstrato chamado **grafo** que é usado para modelar tais situações.

GRAFOS

- **Grafo**: conjunto de vértices e arestas.
- **Vértice**: objeto simples que pode ter nome e outros atributos.
- **Aresta**: conexão entre dois vértices.



- Notação: $G = (V, A)$
  - G: grafo
  - V: conjunto de vértices
  - A: conjunto de arestas

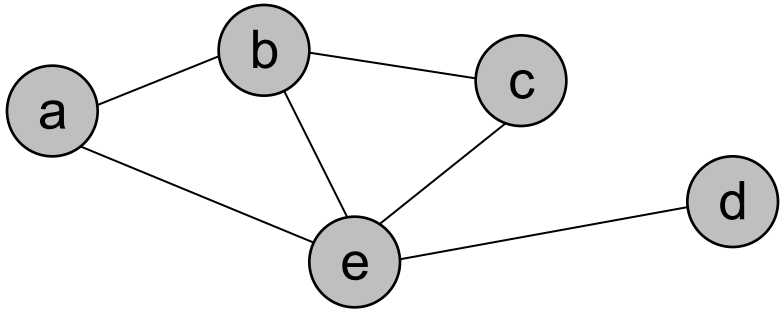
GRAFOS

Aplicações

- Alguns exemplos de problemas práticos que podem ser resolvidos através de uma modelagem em grafos:
  - Ajudar máquinas de busca a localizar informação relevante na Web.
  - Descobrir os melhores casamentos entre posições disponíveis em empresas e pessoas que aplicaram para as posições de interesse.
  - Descobrir qual é o roteiro mais curto para visitar as principais cidades de uma região turística.

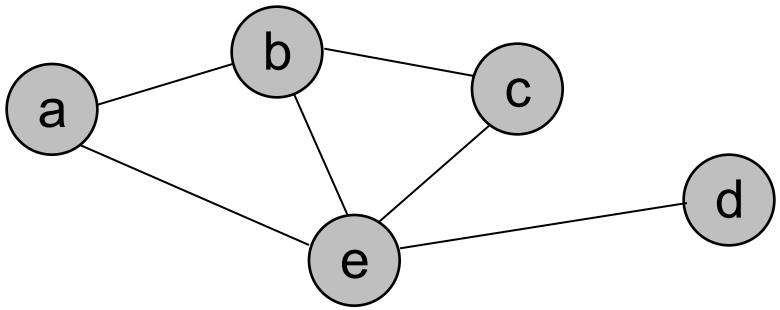
GRAFOS

- Definição:
  - G (V, E), onde:
    - V é um conjunto de vértices (ou nodos)
      - **|V| denota o número de elementos de V**
    - E é uma coleção de pares de V, chamados de aresta (ou arco)
      - **|E| denota o número de elementos de E**
  - As arestas descrevem relações entre os vértices
  - Representação Geométrica



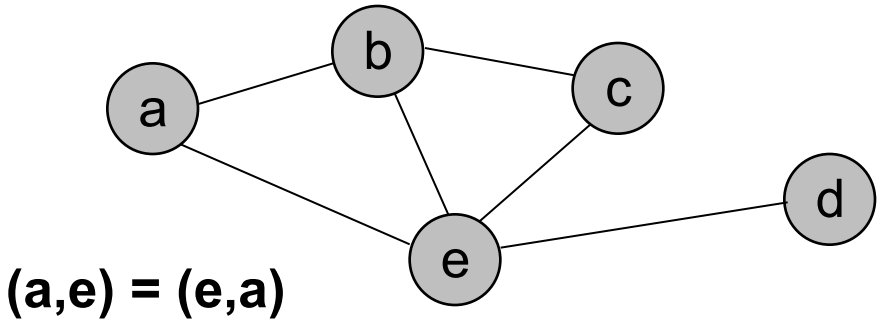
GRAFOS

- Terminologia:
  - Incidência:
    - a aresta *(u,v)* é dita incidente à *u* e a *v*
  - Adjacência: dois vértices *u* e *v* são adjacentes se
    - existe aresta *(u,v)*



GRAFOS

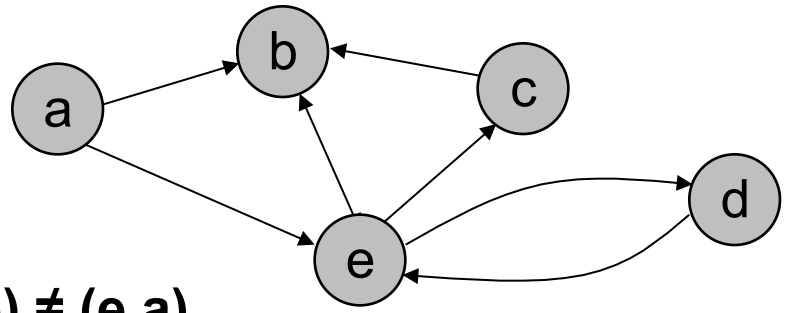
- Grafo não direcionado
  - As arestas (u,v) e (v,u) são consideradas como uma única aresta
  - A relação de adjacência é simétrica
    - Neste caso *(u,v)* é igual de *(v,u)*



(a,e) = (e,a)

GRAFOS

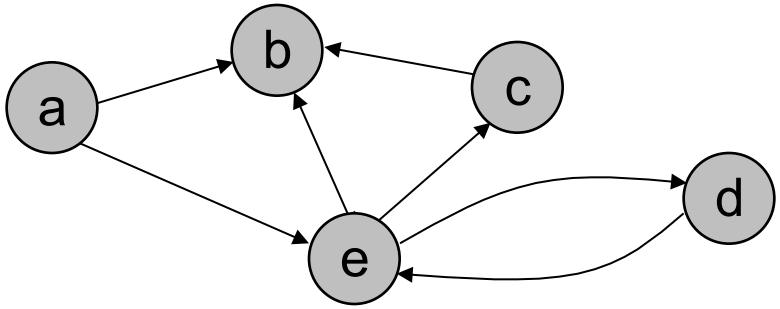
- Grafo Direcionado (ou Dígrafo)
  - Grafo em que as arestas são pares ordenados
  - Neste caso *(u,v)* é diferente de *(v,u)*



(a,e) ≠ (e,a)

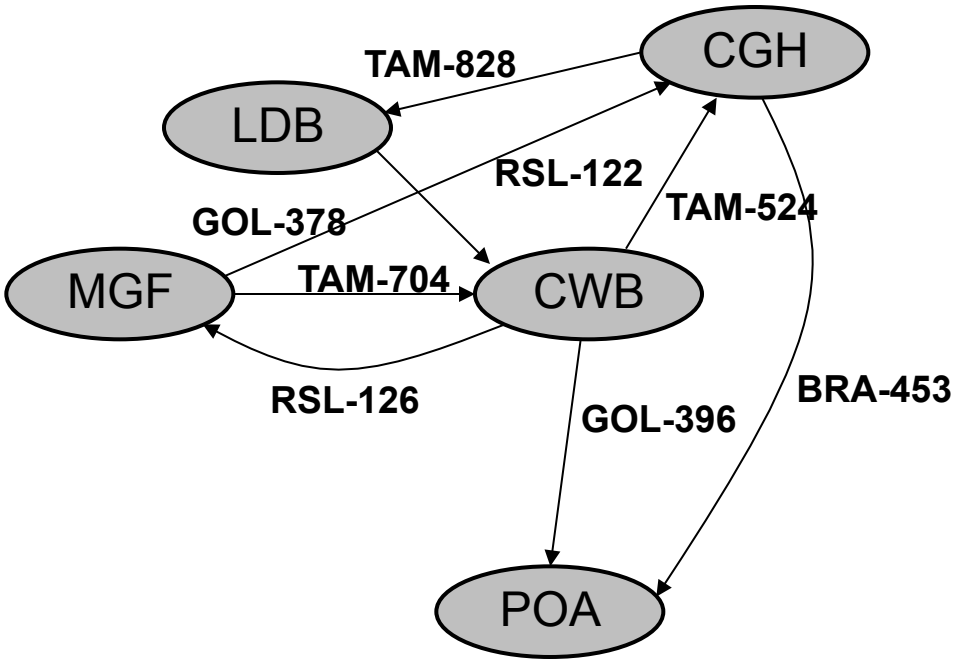
GRAFOS

- Grafo **Direcionado**:
  - Grau de entrada de v:
    - número de arestas convergentes a v
  - Grau de saída de v:
    - número de arestas divergentes de v
  - Exemplo
    - O grau de saída do nó e é 3
    - O grau de entrada do nó e é 2



GRAFOS

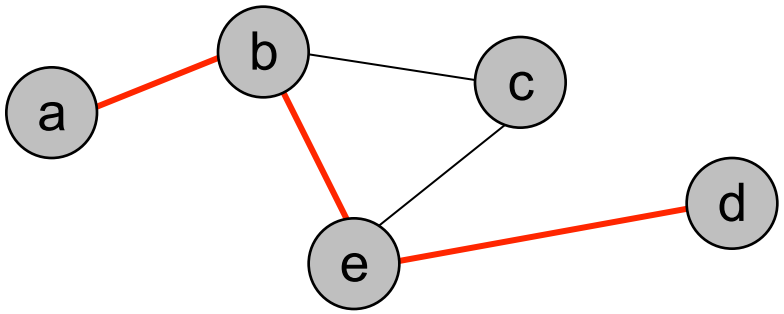
- Grafo Direcionado
  - Exemplo: representação de malha aérea (linhas entre aeroportos)



GRAFOS

- Caminho: um caminho do vértice v_1 ao vértice v_k é uma sequência de vértices $v_1 ... v_k$ tal que (v_j, v_{j+1}) pertence a E

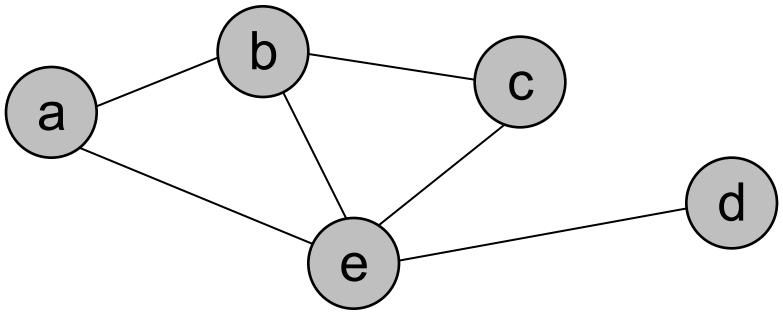
- Comprimento de um caminho: número de arestas percorridas no caminho
  - *d, e, b, a* é um caminho de comprimento 3



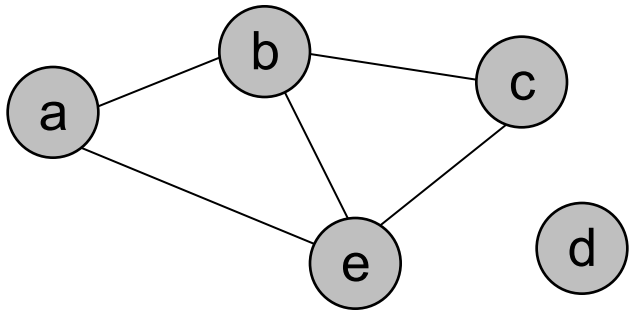
GRAFOS

- Grafo conexo: grafo que possui caminho entre cada par de vértices de V

- Grafo desconexo: grafo não conexo



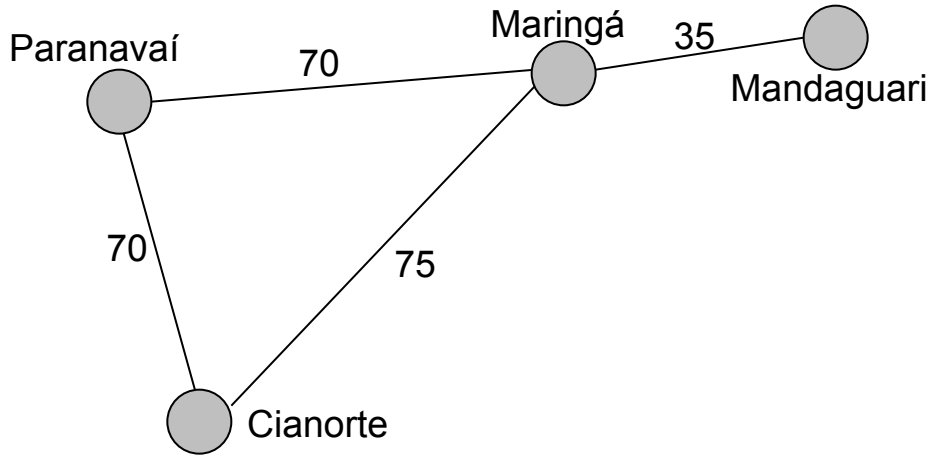
Conexo



Desconexo

GRAFOS

- Grafo ponderado: grafo em que se associa um valor (ou peso) a cada aresta
  - O peso de uma aresta *e* é denotado por *w(e)*

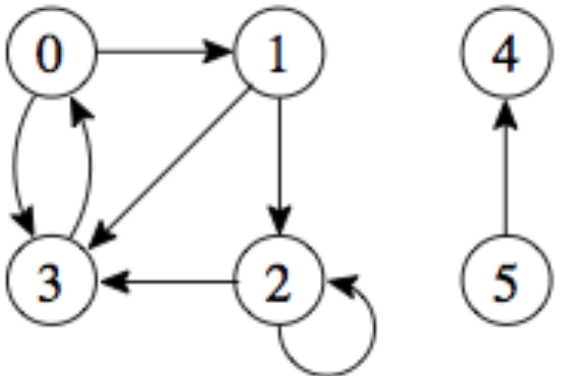


- *w(Maringá, Paranavaí)=70*

GRAFOS

- Ciclo em grafo direcionado

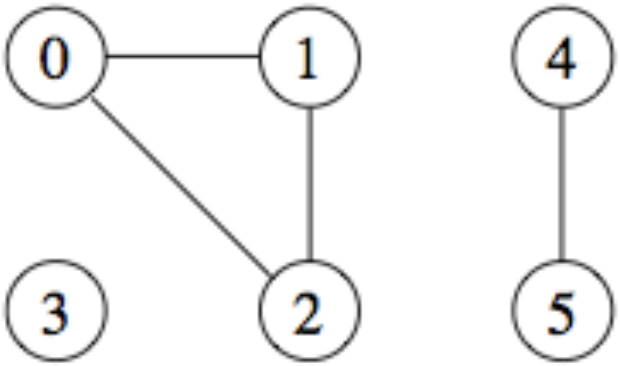
Ex.: O caminho $(0, 1, 2, 3, 0)$ forma um ciclo. O caminho$(0, 1, 3, 0)$ forma o mesmo ciclo que os caminhos $(1, 3, 0, 1)$ e $(3, 0, 1, 3)$.



GRAFOS

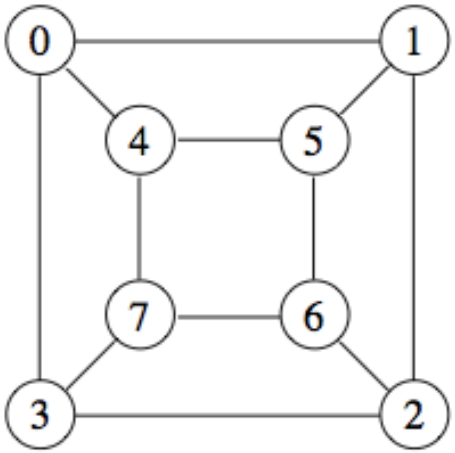
- Ciclo em um grafo não direcionado

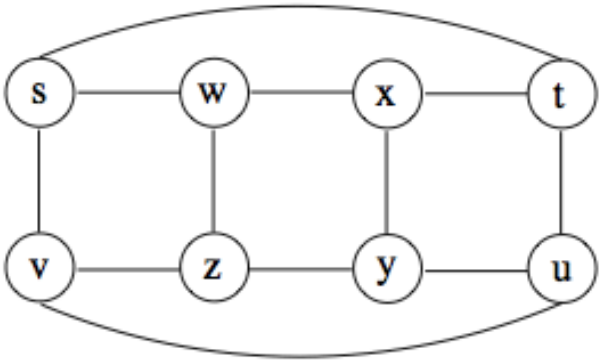
Ex.: O caminho $(0, 1, 2, 0)$ é um ciclo.



Grafos Isomorfos

- $G = (V, A)$ e $G' = (V', A')$ são isomorfos se existir uma bijeção $f : V \to V'$ tal que $(u, v) \in A$ se e somente se $(f(u), f(v)) \in A'$.

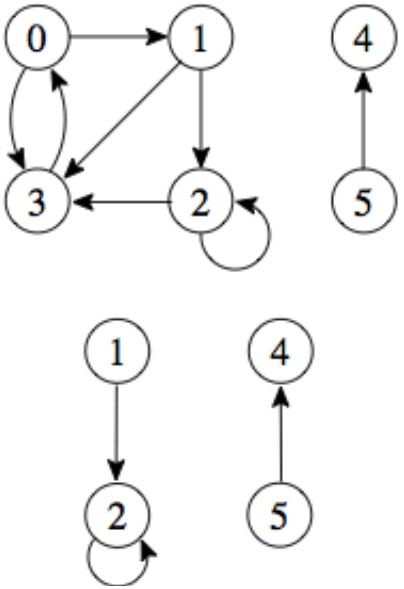




Subgrafos

- Um grafo $G' = (V', A')$ é um subgrafo de $G = (V, A)$ se $V' \subseteq V$ e $A' \subseteq A$.
- Dado um conjunto $V' \subseteq V$, o subgrafo induzido por V' é o grafo $G' = (V', A')$, onde $A' = \{(u, v) \in A | u, v \in V'\}$.

Ex.: Subgrafo induzido pelo conjunto de vértices $\{1, 2, 4, 5\}$.



O Tipo Abstratos de Dados Grafo

- Importante considerar os algoritmos em grafos como **tipos abstratos de dados**.
- Conjunto de operações associado a uma estrutura de dados.
- Independência de implementação para as operações.

Operadores do TAD Grafo

1. *FGVazio(Grafo)*: Cria um grafo vazio.
2. *InsereAresta(V1,V2,Peso, Grafo)*: Insere uma aresta no grafo.
3. *ExisteAresta(V1,V2,Grafo)*: Verifica se existe uma determinada aresta.
4. Obtem a lista de vértices adjacentes a um determinado vértice (tratada a seguir).
5. *RetiraAresta(V1,V2,Peso, Grafo)*: Retira uma aresta do grafo.
6. *LiberaGrafo(Grafo)*: Liberar o espaço ocupado por um grafo.
7. *ImprimeGrafo(Grafo)*: Imprime um grafo.
8. *GrafoTransposto(Grafo,GrafoT)*: Obtém o transposto de um grafo direcionado.
9. *RetiraMin(A)*: Obtém a aresta de menor peso de um grafo com peso nas arestas.

Operação "Obter Lista de Adjacentes"

1. *ListaAdjVazia(v, Grafo)*: retorna *true* se a lista de adjacentes de v está vazia.
2. *PrimeiroListaAdj(v, Grafo)*: retorna o endereço do primeiro vértice na lista de adjacentes de v.
3. *ProxAdj(v, Grafo, u, Peso, Aux, FimListaAdj)*: retorna o vértice u (apontado por *Aux*) da lista de adjacentes de v, bem como o peso da aresta (v, u). Ao retornar, *Aux* aponta para o próximo vértice da lista de adjacentes de v, e *FimListaAdj* retorna *true* se o final da lista de adjacentes foi encontrado.

Implementação da Operação "Obter Lista de Adjacentes"

- É comum encontrar um pseudo comando do tipo:

  for u ∈ ListaAdjacentes (v) **do** { faz algo com u }

- O trecho de programa abaixo apresenta um possível refinamento do pseudo comando acima.

```
if not ListaAdjVazia(v, Grafo)
then begin
      Aux := PrimeiroListaAdj(v, Grafo);
      FimListaAdj := false;
      while not FimListaAdj
      do ProxAdj(v, Grafo, u, Peso, Aux, FimListaAdj);
      end;
```

Implementação de grafos

- Grafos são estruturas de dados presentes em Ciência da Computação, e os algoritmos para trabalhar com eles são Fundamentais na área.
- Existem muitos **problemas** interessantes definidos em termos de grafos
  - Caminho entre dois elementos;
  - Caminho Mínimo;
  - Otimização de recursos limitados;
  - ...
- **Como podemos representar um grafo em um computador?**

Implementação de grafos

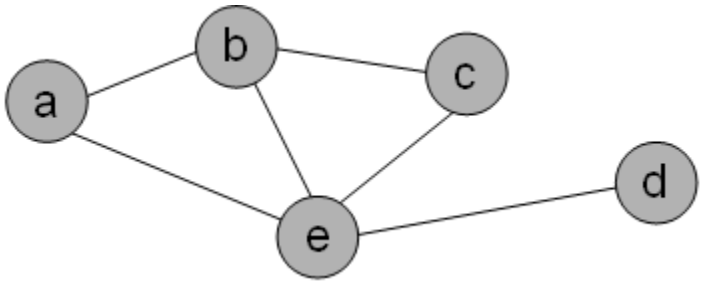
- Representações Usuais:
  - Matrizes
    - de adjacências
    - de incidências
  - Lista de adjacências

Matriz de adjacências

- Para um grafo G = (V,E):
  - Cria-se uma **matriz M** com dimensões |V| x |V|;
  - Cada linha *i* e cada coluna *j* é associada à um vértice do grafo:
    - M[i,j]=1 se **existe** uma aresta do vértice *i* para *j*
    - M[i,j]=0 se **não existe** uma aresta do vértice *i* para *j*;

Exemplo:

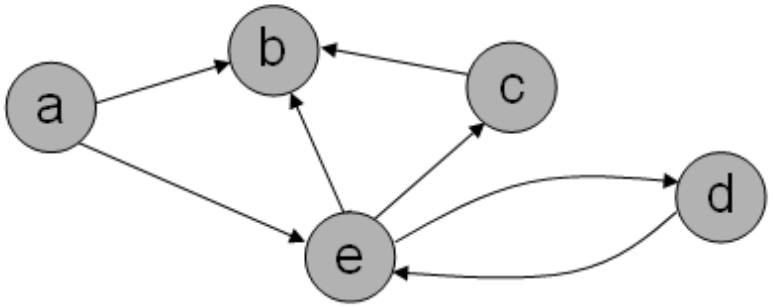
Gafo não direcionado



	a	b	c	d	e
a	0	1	0	0	1
b	1	0	1	0	1
c	0	1	0	0	1
d	0	0	0	0	1
e	1	1	1	1	0

Matriz de adjacências

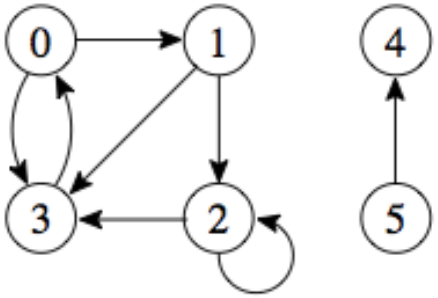
- Observe que:
  - Em grafo **não direcionado** a **matriz é simétrica**
  - Em grafo **direcionado** a **matriz é não-simétrica**
- Exemplo:
  - Grafo Dígrafo M[i,j] ≠ M[j,i]

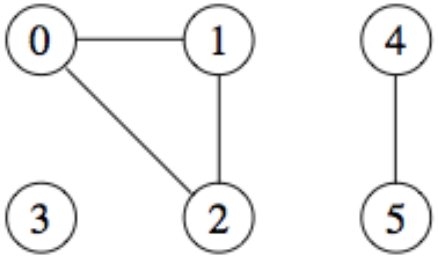


	a	b	c	d	e
a	0	1	0	0	1
b	0	0	0	0	0
c	0	1	0	0	0
d	0	0	0	0	1
e	0	1	1	1	0

M[b,a] ≠ M[a,b]

Matriz de adjacências





	0	1	2	3	4	5
0		1		1		
1			1	1		
2			1	1		
3	1					
4						
5						

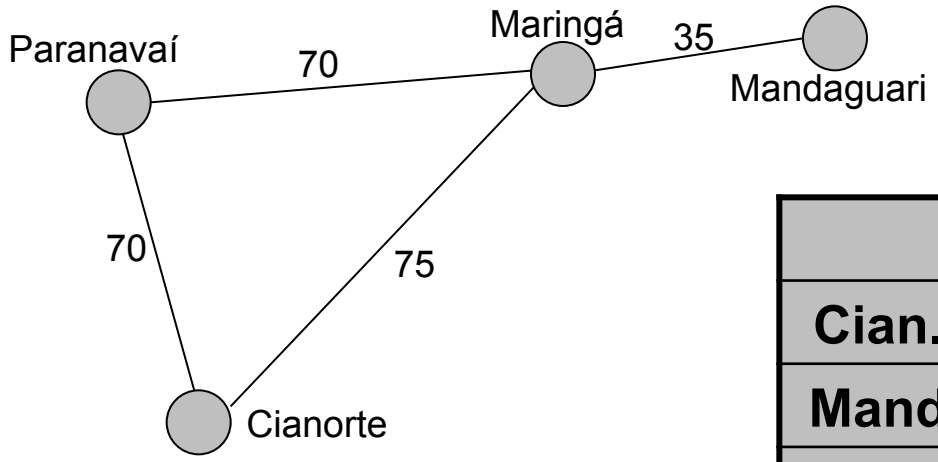
(a)

	0	1	2	3	4	5
0		1	1			
1	1		1			
2	1	1				
3						
4						
5						

(b)

Matriz de adjacências

- Grafo ponderado:
  - Coloca-se em M[i,j] o rótulo ou o peso associado à aresta



	Cian	Mand	Mga	Pvai
Cian.	0	0	75	70
Mand	0	0	35	0
Mga	75	35	0	70
Pvai	70	0	70	0

Matriz de adjacências

- Definição da estrutura tipo grafo implementado com matriz de adjacências

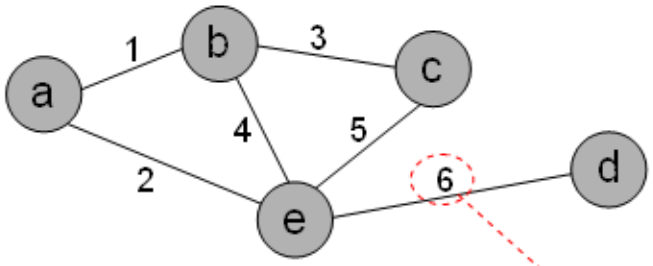
```
const MaxNumVertices = 100;
      MaxNumArestas  = 4500;
type
  TipoValorVertice = 0..MaxNumVertices;
  TipoPeso         = integer;
  TipoGrafo = record
                Mat: array[TipoValorVertice, TipoValorVertice]
                     of TipoPeso;
                NumVertices: 0..MaxNumVertices;
                NumArestas : 0..MaxNumArestas;
              end;
```

Matriz de adjacências

- **Vantagens**:
  - Pode ser preferível quando o grafo é **denso**
    - **$|E|$ é próximo de $|V|^2$**
  - Possibilidade de acesso rápido à informação sobre uma aresta
    - Tempo de acesso independente de $|V|$ ou $|A|$
  - Custo constante para inserção ou retirada de um vértice
- **Desvantagens**:
  - ocupa $|V|^2$ posições na memória
  - ler ou examinar a matriz tem complexidade de tempo $O(|V|^2)$

Matriz de incidências

- Outra forma de representar:
- Para um grafo G = (V,E):
  - Cria-se uma matriz M com dimensões |V| x |E|
  - Cada linha *i* (vértice) e cada coluna *j* (aresta) do grafo
    - M[i,j]=1 se j incide em i;
    - M[i,j]=0 caso contrário;
- Exemplo:

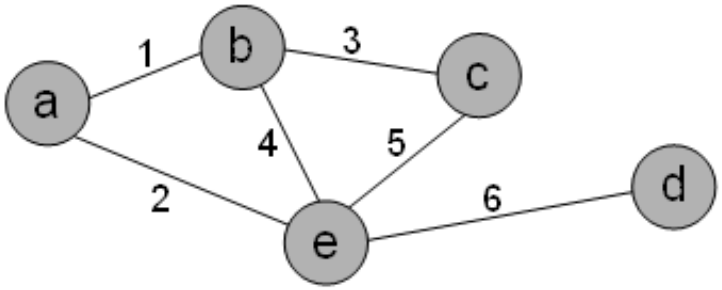


	1	2	3	4	5	6
a	1	1	0	0	0	0
b	1	0	1	1	0	0
c	0	0	1	0	1	0
d	0	0	0	0	0	1
e	0	1	0	1	1	1

Neste caso estes números não são pesos, são identificadores das arestas.

Matriz de incidências

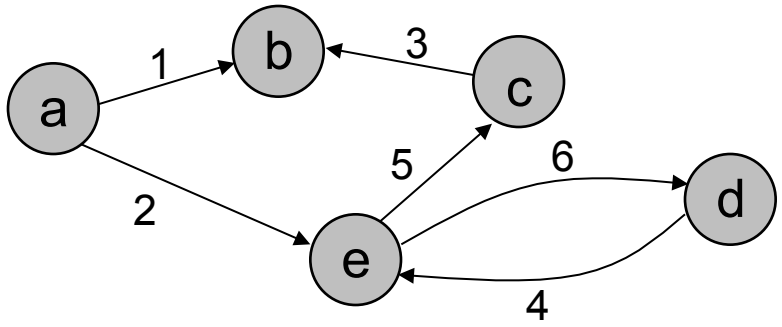
- Em cada coluna dois 1's;
- Em cada linha o número de 1's é igual ao grau do vértice;
- Desvantagem:
  - é preciso conhecer antecipadamente o número de vértices e arestas do grafo;



	1	2	3	4	5	6
a	1	1	0	0	0	0
b	1	0	1	1	0	0
c	0	0	1	0	1	0
d	0	0	0	0	0	1
e	0	1	0	1	1	1

Matriz de incidências

- Matriz de incidências para grafo dígrafo
  - -1 para arestas divergentes (saída)
  - 1 para arestas convergentes (entrada)



	1	2	3	4	5	6
a	-1	-1	0	0	0	0
b	1	0	1	0	0	0
c	0	0	-1	0	1	0
d	0	0	0	-1	0	1
e	0	1	0	1	-1	-1

Exemplo:

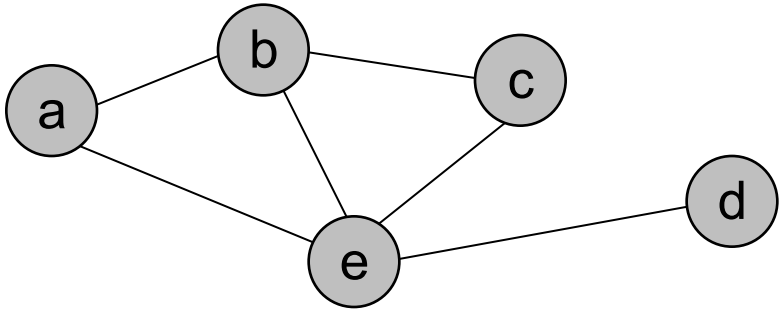
Nó e tem as arestas de saída 5 e 6

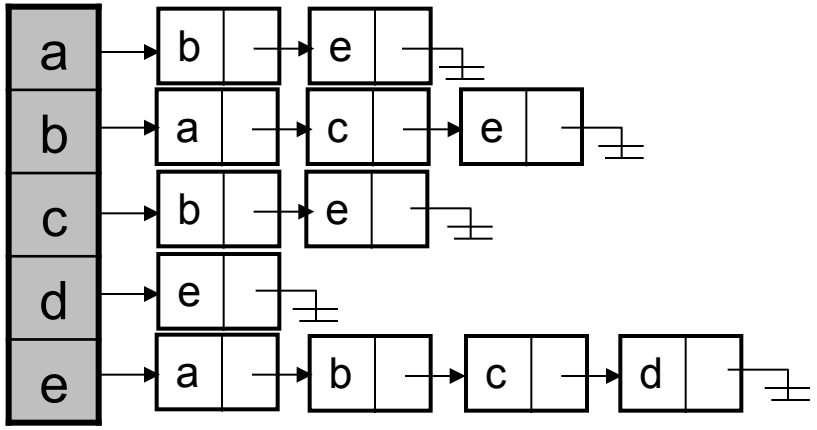
Nó e tem as arestas de entrada 2 e 4

Listas de Adjacência usando apontadores

Lista de adjacências

- Lista de adjacências consiste em um arranjo *Adj* com |V| listas encadeadas;
- Em cada vértice, a lista contém ponteiros para todos os demais vértices que são adjacentes a ele (ordem arbitrária);
- Exemplo:

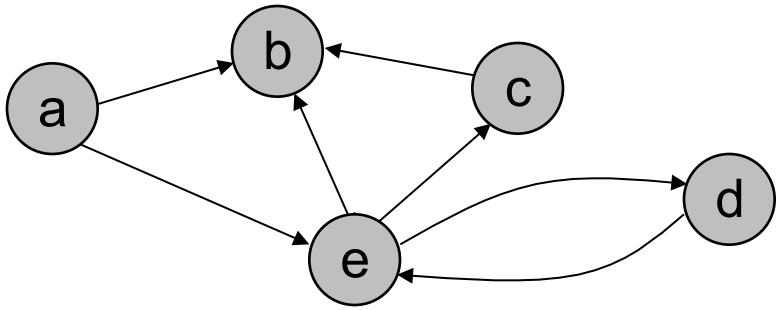


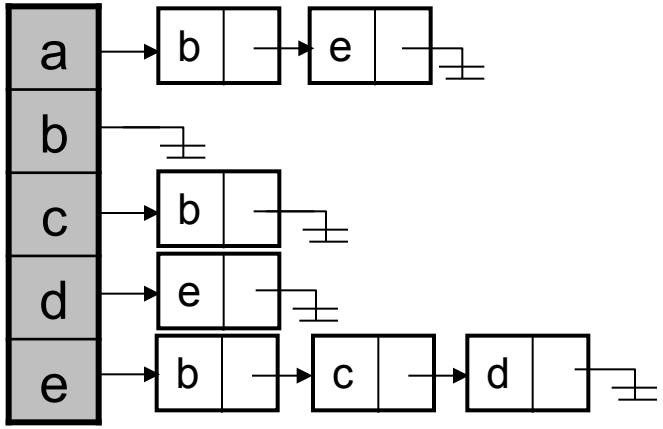


  - Complexidade de espaço: |V| + 2|E|
    - aresta não orientada aparece duas vezes na lista de adjacências.

Lista de adjacências

- Lista de adjacências para dígrafos
  - Exemplo:

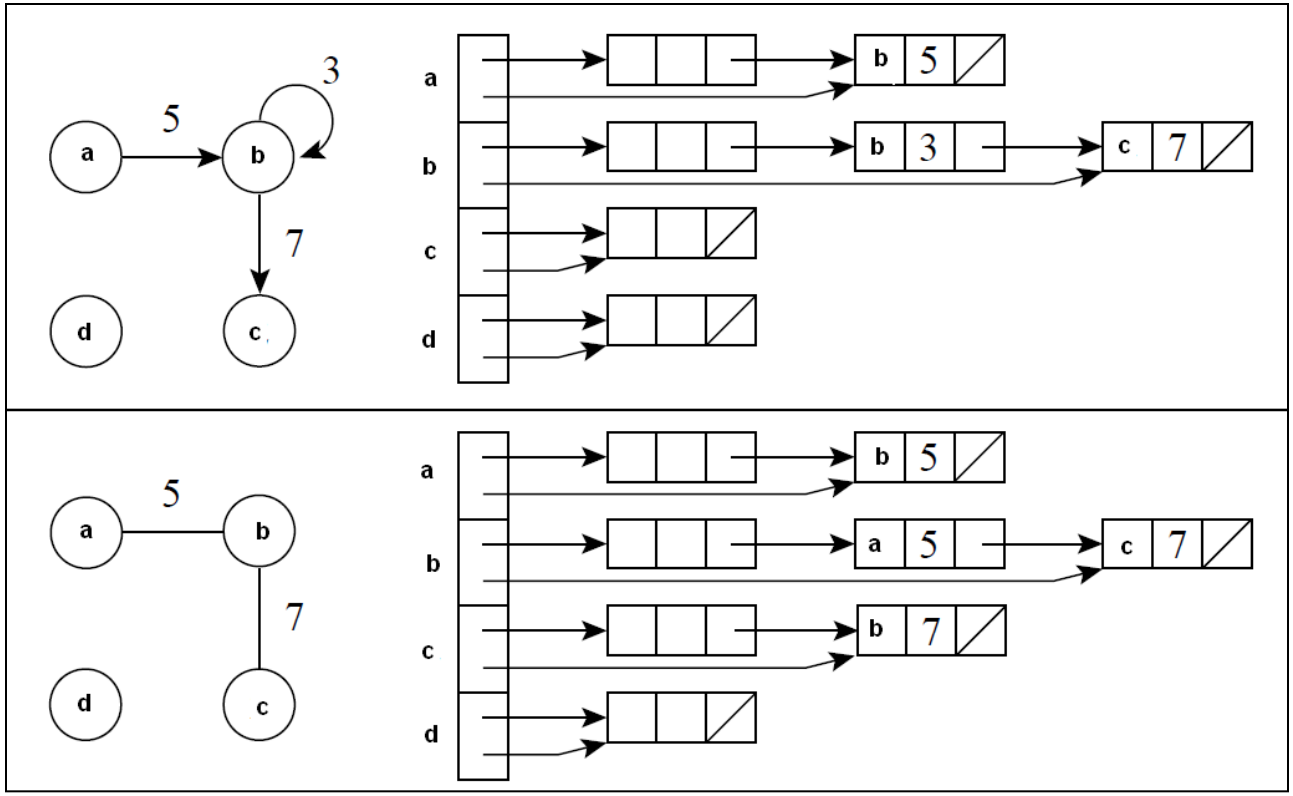




  - Complexidade de espaço: |V| + |E|
    - A soma dos comprimentos de todas as listas de adjacências é |E|

Lista de adjacências

- Podem ser adaptadas para admitir outras variantes de grafos
  - Exemplo, grafos ponderados:



Vetor de registro com dois elementos: primeiro/ultimo

Lista de adjacências

- Indicada para grafos esparsos
  - $|E|$ é muito menor do que $|V|^2$
- Representação compacta e usada na maioria das implementações
- Desvantagem
  - Pode ter tempo $O(|V|)$ para determinar se existe uma aresta entre o vértice *i* e *j*
    - pode existir $O(|V|)$ vértices na lista de adjacentes do vértice i

Problema clássico em grafos

- Menor caminho (origem única)
  - para grafos sem pesos:
    - busca em amplitude (largura);
  - Para grafos ponderados e sem pesos negativos:
    - algoritmos de Dijkstra;

Busca em Largura

- Base para outros algoritmos em grafos
  - Algoritmo Prim (árvore geradora mínima)
  - Algoritmo Dijkstra (caminho mais curto)
- Encontra o menor caminho a partir de um vértice inicial
- Dado um grafo G(V,E) e um vértice origem, o algoritmo descobre todos os vértices a uma distância *k* do vértice origem antes de descobrir qualquer vértice a uma distância k + 1.

Busca em Largura

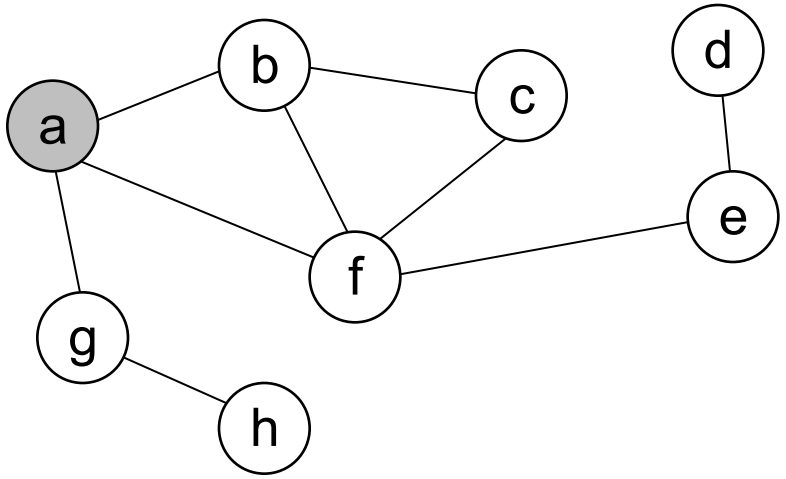
– Método:

1. Inicialmente marca-se o vértice de origem como visitado;
2. Coloca-se todos os seus vértices adjacentes ainda não visitados em uma fila F;
3. Retira-se o primeiro vértice da fila, marcando-o como visitado.

- Repete-se o processo a partir do passo 2, até que não haja vértices não visitados.

Busca em Largura

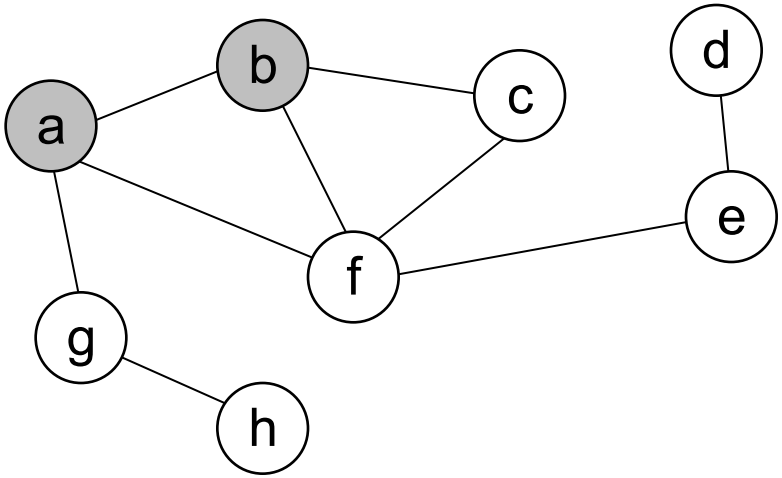
- Exemplo
  - Vértice de origem: a



Fila	b	f	g

Busca em Largura

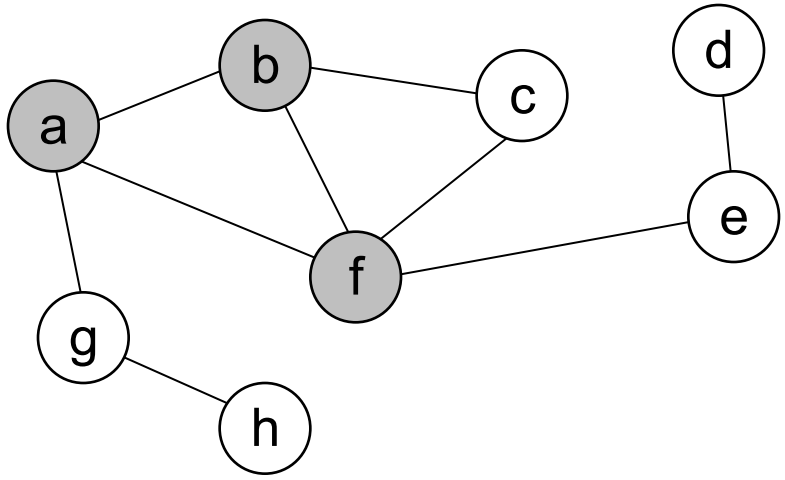
- Exemplo



Fila	f	g	**c**

Busca em Largura

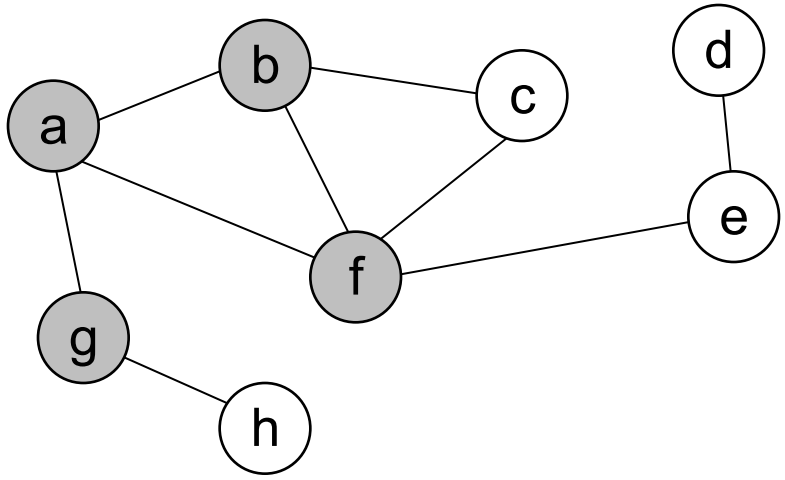
- Exemplo



Fila	g	**c**	**e**

Busca em Largura

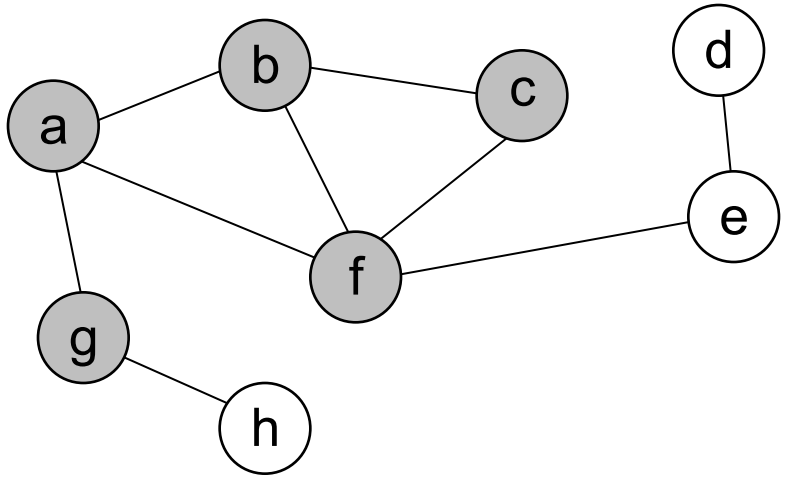
- Exemplo



Fila	c	e	**h**

Busca em Largura

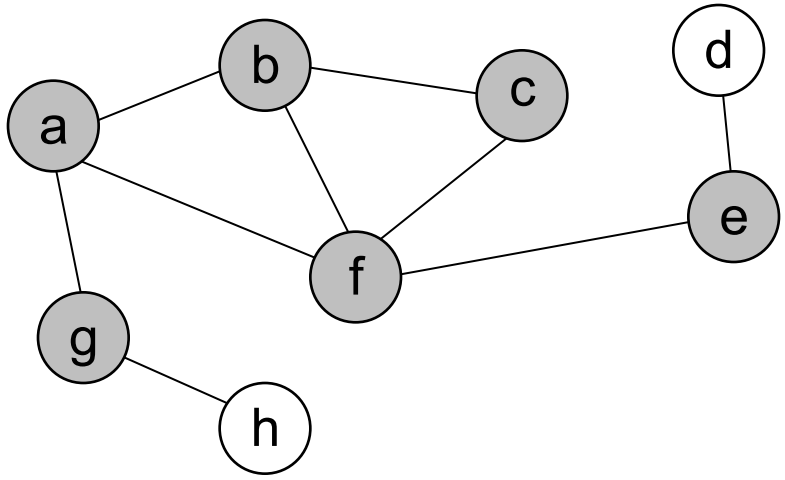
- Exemplo



Fila	e	h	

Busca em Largura

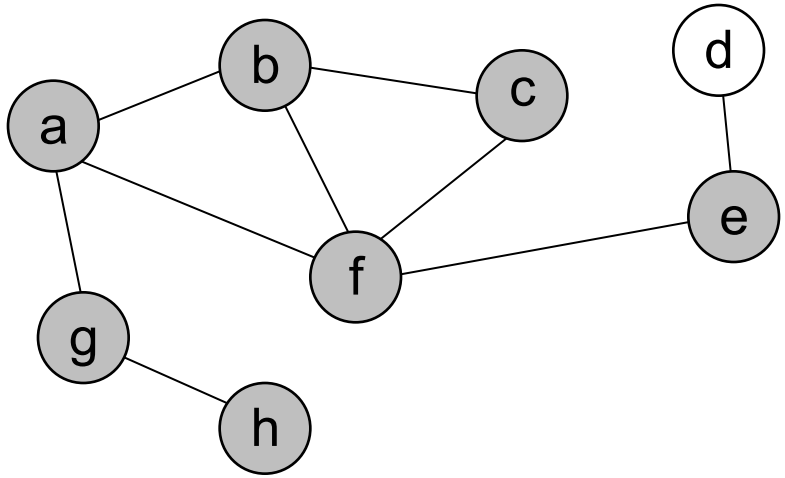
- Exemplo



Fila	h	d	

Busca em Largura

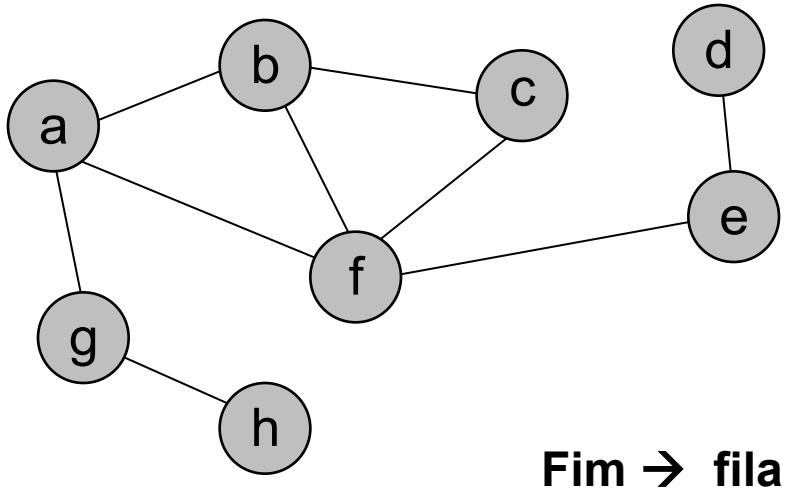
- Exemplo



Fila	d		

Busca em Largura

- Exemplo



Fim → fila vazia e todos os vértices visitados.

Fila			

Exercícios

1. Implemente em uma linguagem de alto nível a estrutura de um grafo utilizando uma matriz de incidência. O algoritmo deve:
   - Ler os vértices e arestas informados pelo usuário e informar:
   - Grafo orientado ou não?
   - Possui ciclos?
   - Possui nós isolados?
   - É conexo?
   - Qual o grau de cada nó do grafo?

2. Faça o mesmo exercício anterior usando lista de adjacência.

Referencias

- ZIVIANI, Nivio. **Projeto de Algoritmos com implementações em Pascal e C.** São Paulo: Pioneira Thomson Learning, 2a. ed., 2004.
- CORMEN, Leiserson, Rivest e Stein. Algoritmos. Editora Campus, 2002;
- DASGUPTA, Sanjoy et al. **Algoritmos**. Editora: São Paulo: McGraw-Hill, 2009.
- MORAES, Celso Roberto. Estruturas de Dados e Algoritmos. Editora Berkley, 2001.
- Notas de aula prof. Yandre Maldonado. Disponível em: http://www.din.uem.br/yandre/aed.htm