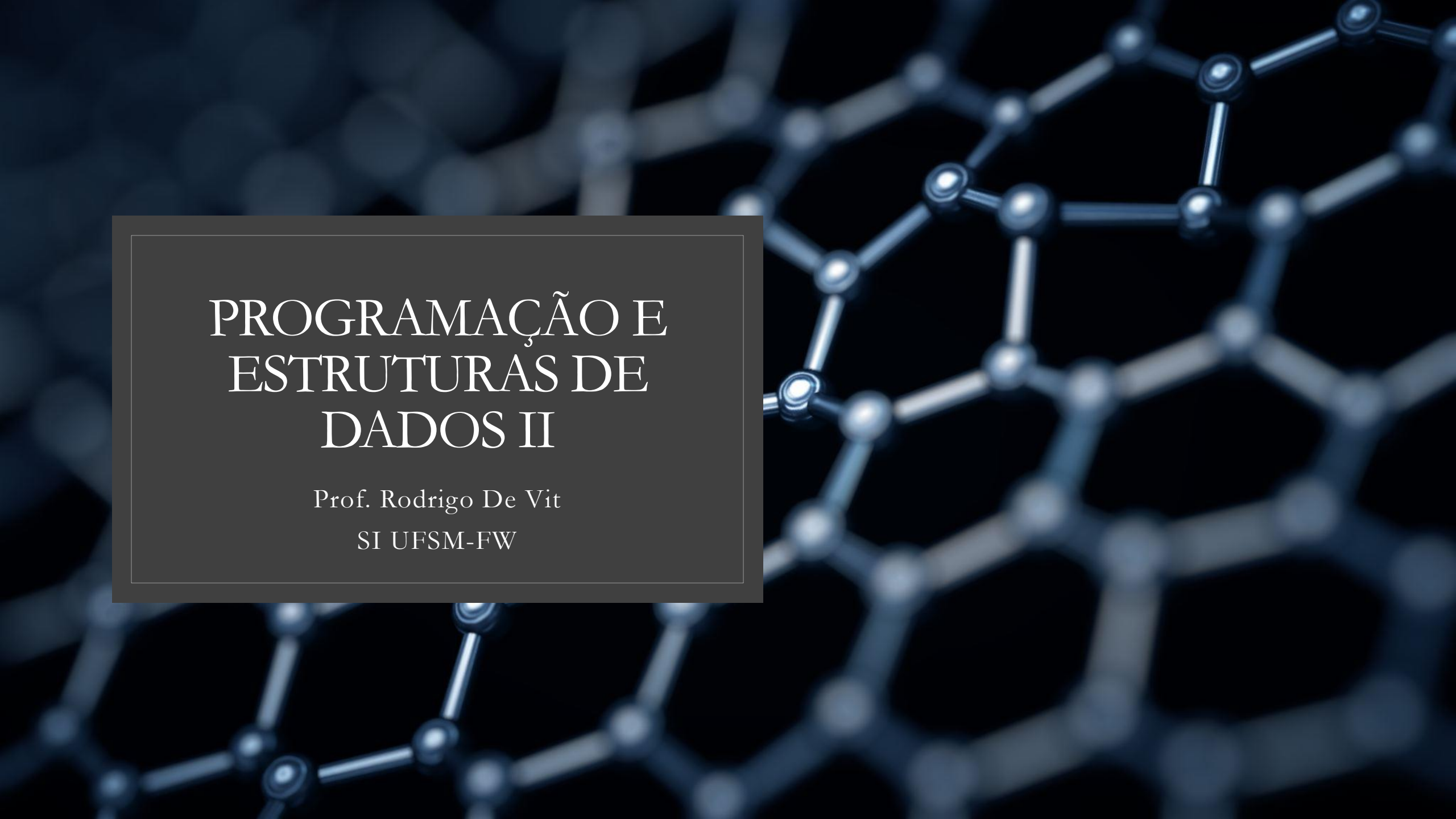
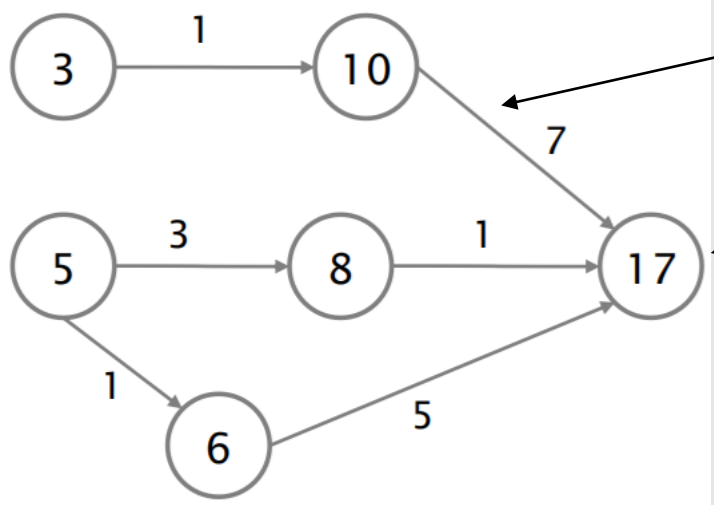


PROGRAMAÇÃO E ESTRUTURAS DE DADOS II

Prof. Rodrigo De Vit

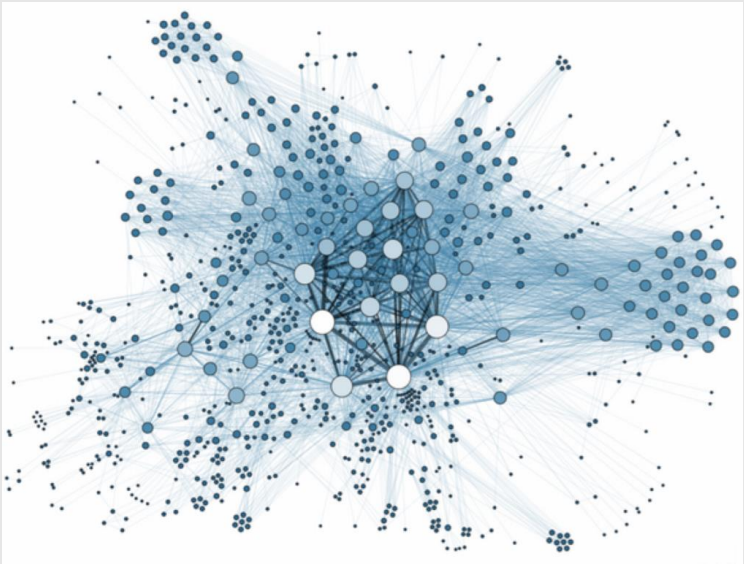
SI UFSM-FW

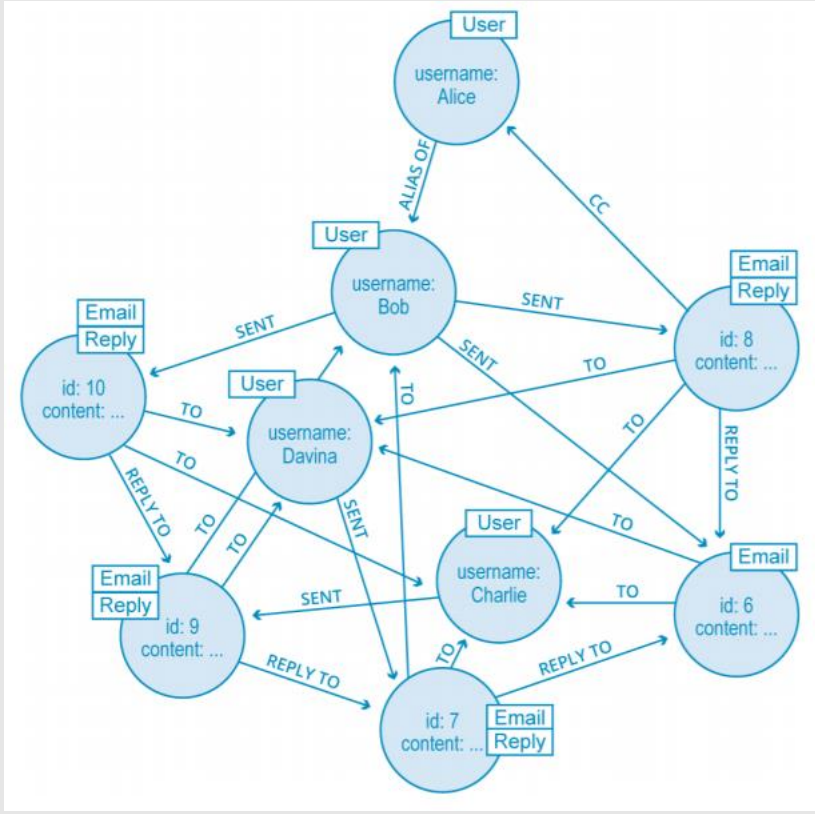
Grafos



Aresta

Vértice





Aplicações de Grafos

grafo	vértices	arestas
Cronograma	tarefas	restrições de preferência
Malha viária	interseções de ruas	ruas
Rede de água (telefônica,...)	Edificações (telefones,...)	Canos (cabos,...)
Redes de computadores	computadores	linhas
Software	funções	chamadas de função
Web	páginas Web	links
Redes Sociais	pessoas	relacionamentos
...		

Grafo não dirigido

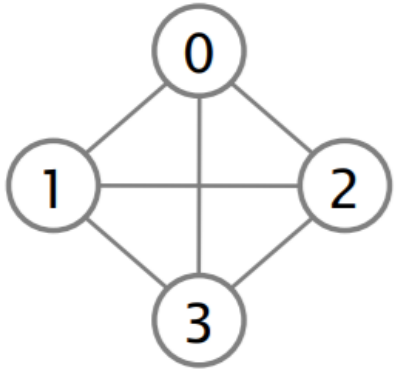
Um *grafo não dirigido* é um par G = (V,E), onde

V é um conjunto de *nós* ou *vértices* e

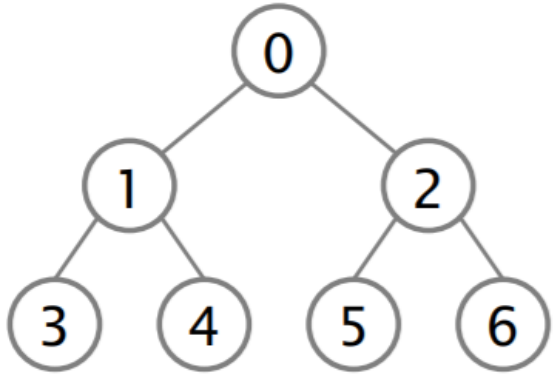
E é um conjunto de *arestas*

uma *aresta* é um conjunto de 2 vértices

Exemplos



vértices: V = {0,1,2,3}
arestas: E = {{0,1},{0,2},{0,3},{1,2},{1,3},{2,3}}



vértices : V = {0,1,2,3,4,5,6}
arestas: E = {{0,1},{0,2},{1,3},{1,4},{2,5},{2,6}}

Grafo dirigido (orientado, ou Digrafo)

Um *grafo dirigido* é um par G = (V,E), onde

V é um conjunto de *n nós* ou *vértices* e

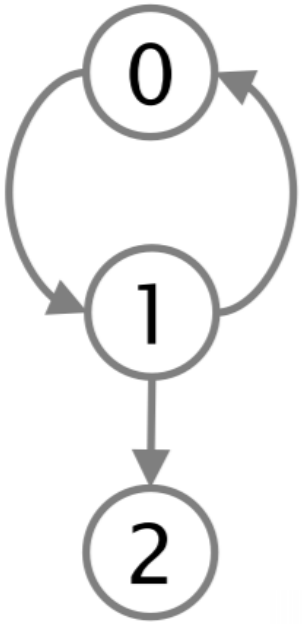
E é um conjunto de *m arcos*

um *arco* é um par ordenado de vértices

Exemplo

vértices: V = {0,1,2}

arcos: E = {(0,1), (1,0), (1,2)}

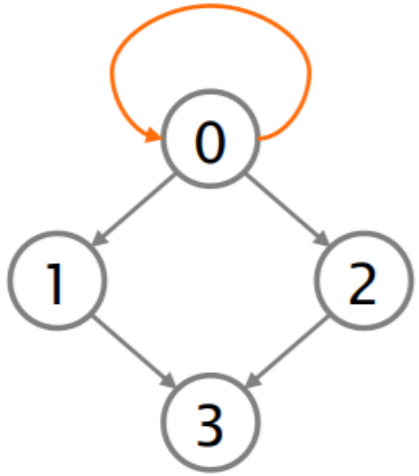


Grafo dirigido (orientado, ou Digrafo)

Exemplo - Digrafo com auto-arco

vértices: V = {0,1,2,3}

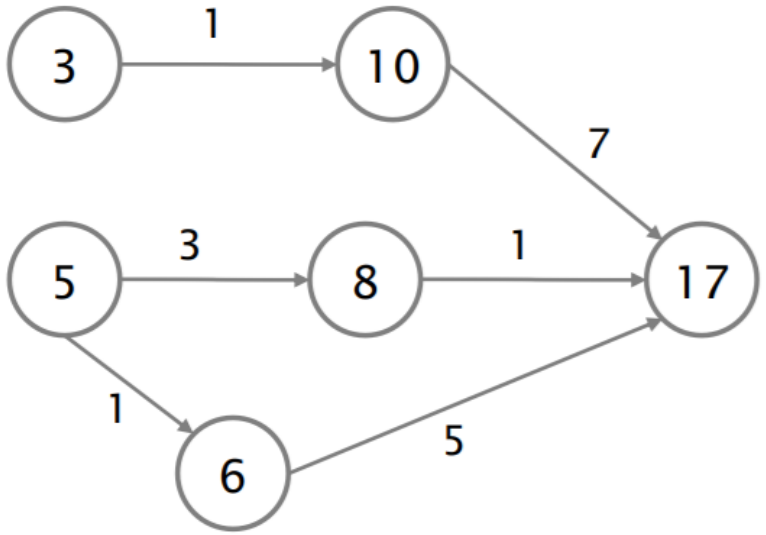
arcos: E = {(0,0), (0,1), (0,2), (1,3), (2,3)}



Grafo ponderado

Um *grafo ponderado* é uma tripla G = (V,E,p), onde
V é um conjunto de *n nós* ou *vértices* e
E é um conjunto de *m arcos*
p é uma função que atribui a cada arco um *peso*

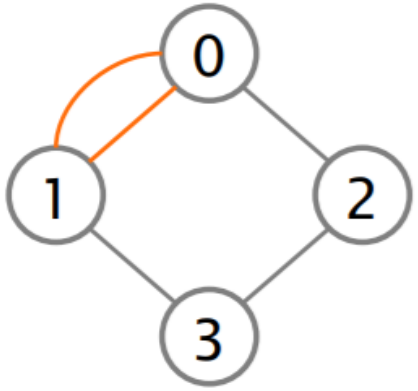
Exemplo



Multigrafo

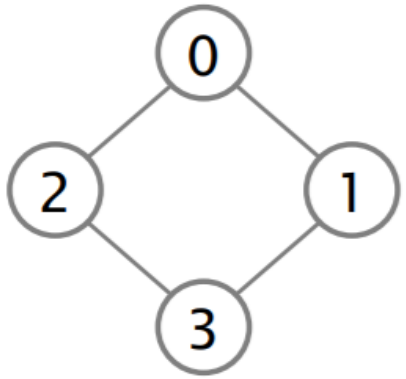
Um *multigrafo* é um grafo onde
dois nós podem estar conectados por mais de uma aresta

Exemplo

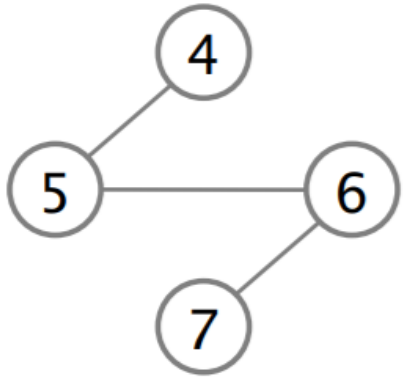


Vértices adjacentes

Vértices conectados por arestas

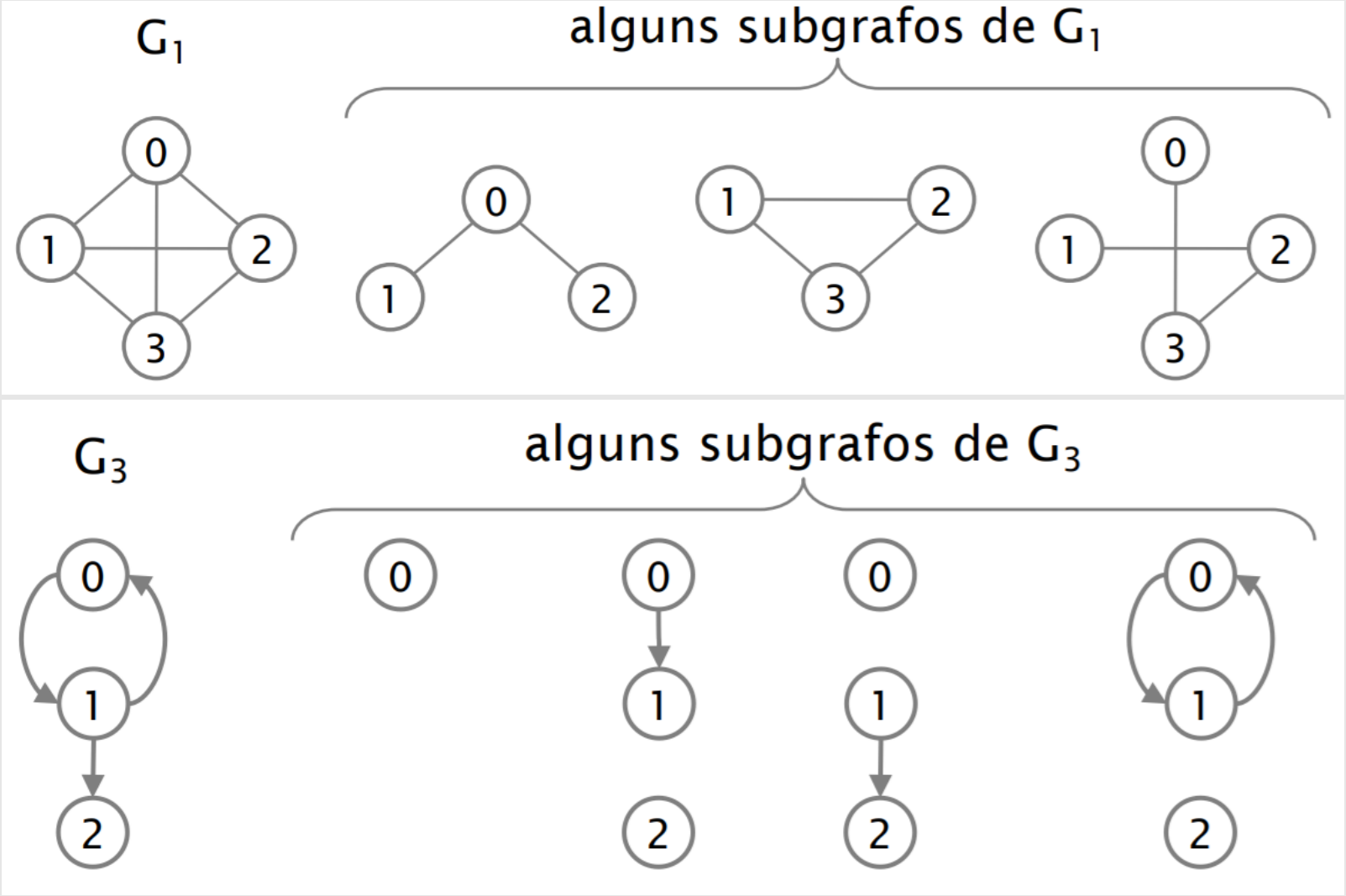


0 e 1
0 e 2
1 e 3
2 e 3



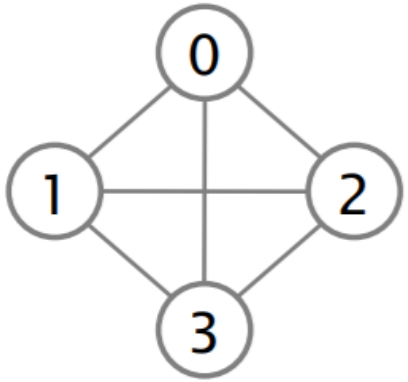
4 e 5
5 e 6
6 e 7

Subgrafo



Grafo completo

Um grafo não direcionado é *completo* sse cada vértice está conectado a cada um dos outros vértices por uma aresta



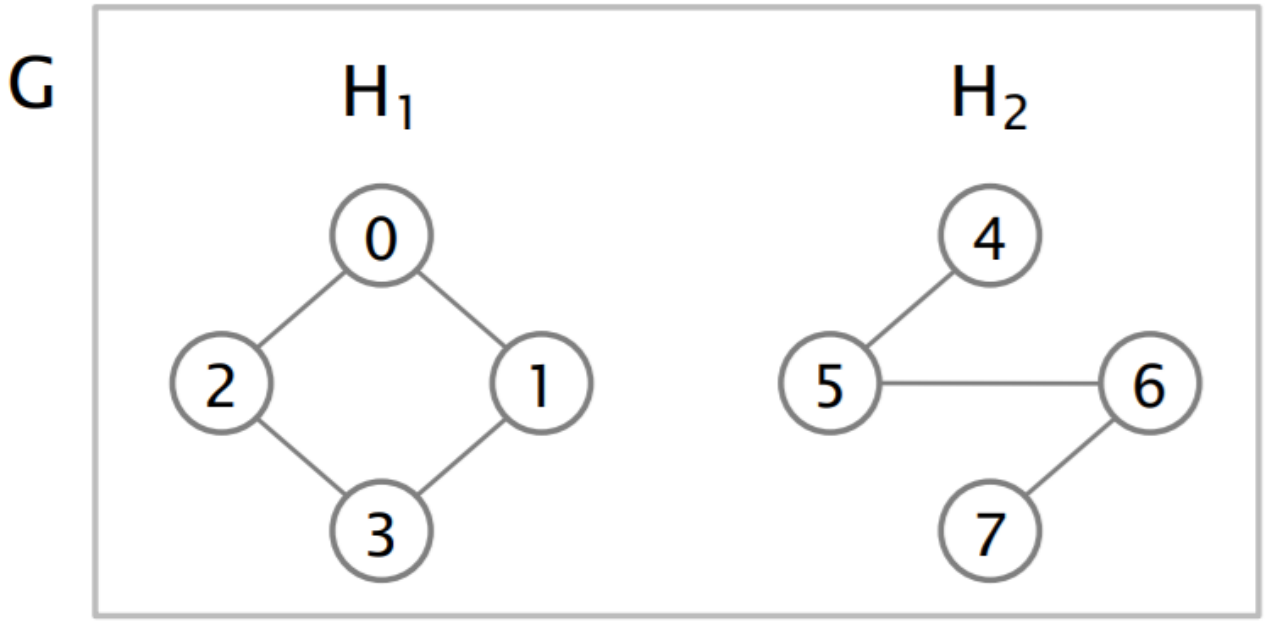
Quantas arestas há em um grafo completo de n vértices?

n (n-1)/2

Grafo conectado

Um grafo não direcionado é *conectado* ou *conexo* sse existe um caminho entre quaisquer dois vértices

Componente conexa de um grafo



Grau

Um vértice possui *grau n* sse
há exatamente *n* arestas incidentes ao vértice

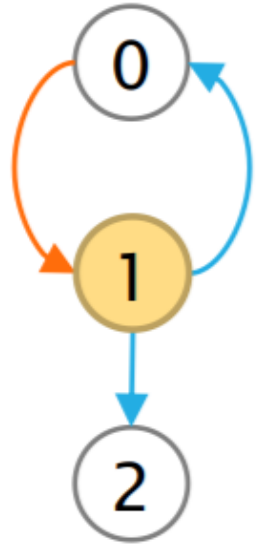
Exemplo:

grau do vértice 1: 3

grau de entrada do vértice 1: 1

grau de saída do vértice 1: 2

G_3



Caminhos

Caminho

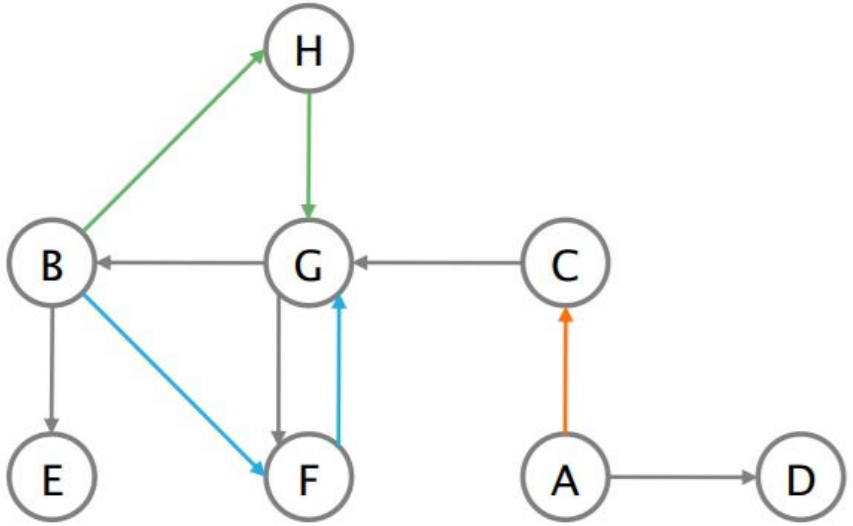
de comprimento 1 entre A e C

de comprimento 2 entre B e G, passando por H

de comprimento 2 entre B e G, passando por F

de comprimento 3 de A a F

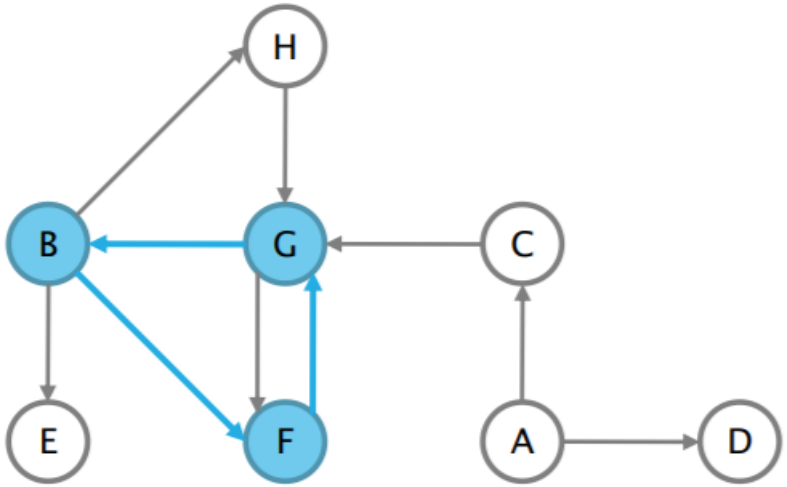
Ciclos



Ciclos

Um *ciclo* é um caminho de um nó para ele mesmo

exemplo: B-F-G-B



Grafo cíclico

contém um ou mais ciclos

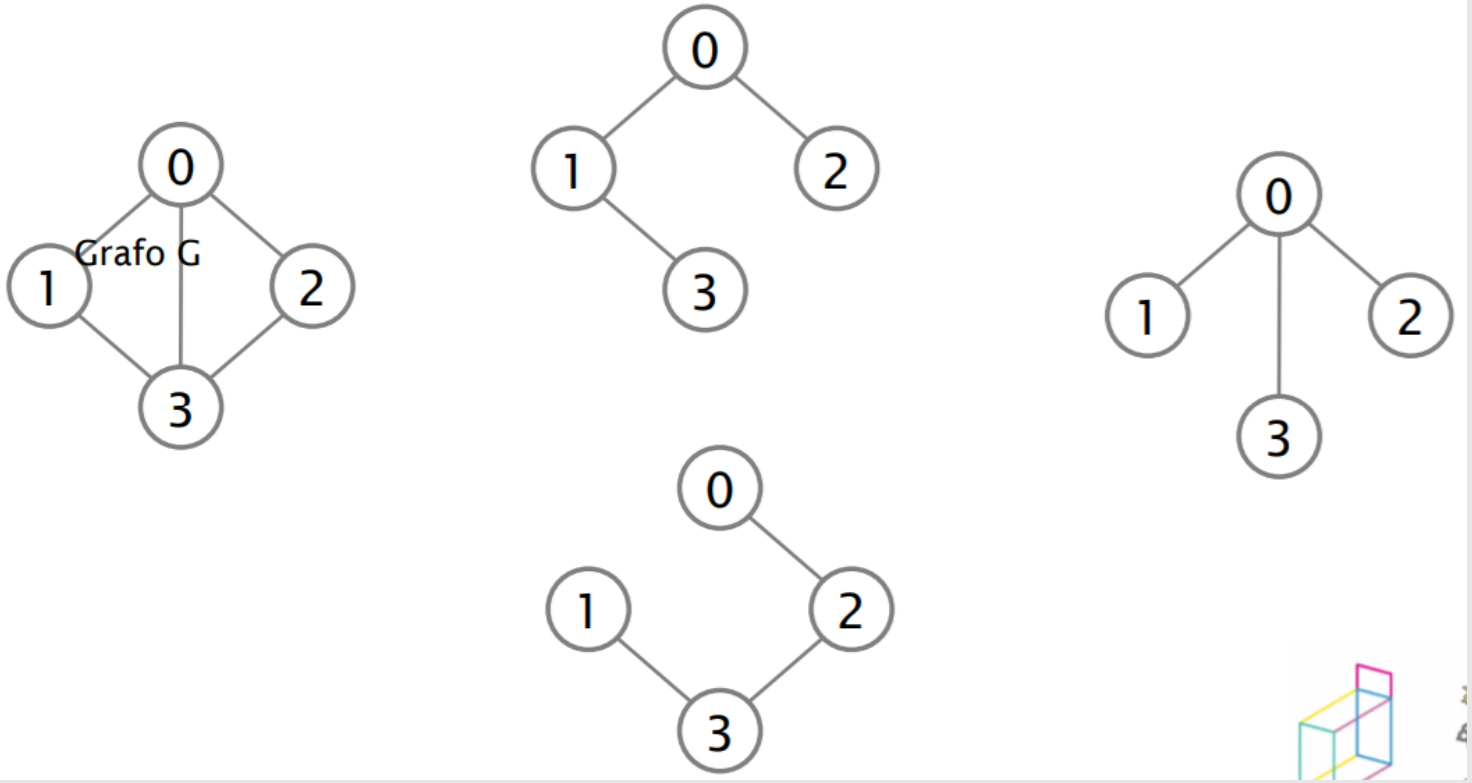
Grafo acíclico

não contém ciclos

em grafos direcionados e não direcionados

Árvore Geradora

subgrafo acíclico contendo todos os vértice com caminhos entre quaisquer 2 vértices

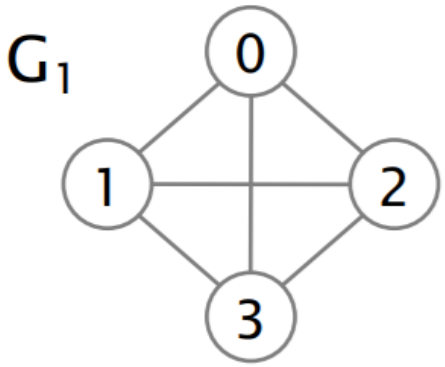


Uma árvore geradora é simplesmente um conjunto de arestas do grafo que gera uma árvore.

De um modo geral, obtém-se uma árvore geradora removendo arestas até eliminar os ciclos, mas mantendo a conexidade. É fácil ver que se um grafo tem n nós, então uma árvore geradora do grafo terá exatamente n — 1 arestas. Observe-se ainda que um grafo pode admitir diferentes árvores geradoras, conforme a escolha de arestas a eliminar.

Representações de grafo - Matriz de adjacências

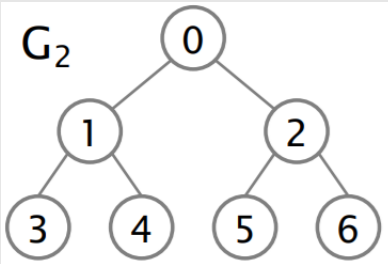
$$mat[i][j] = \begin{cases} 1, \text{ se houver uma aresta do nó i para o nó j} \\ 0, \text{ caso contrário} \end{cases}$$



	0	1	2	3
0	0	1	1	1
1	1	0	1	1
2	1	1	0	1
3	1	1	1	0

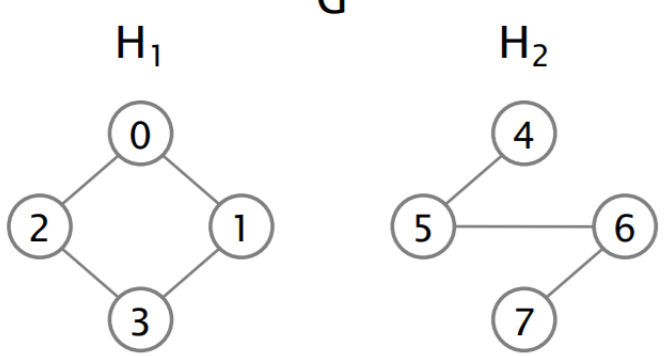
matrizes simétricas para grafos não direcionados

Representações de grafo - Matriz de adjacências



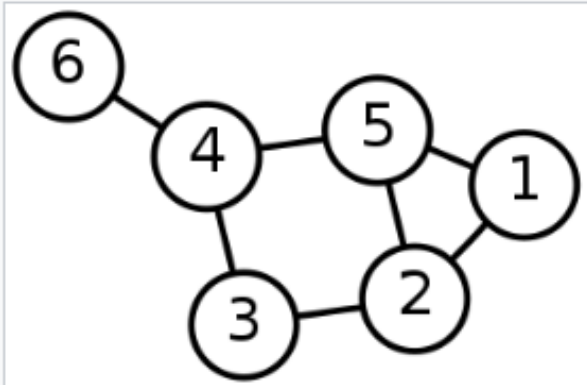
	0	1	2	3	4	5	6
0	0	1	1	0	0	0	0
1	1	0	0	1	1	0	0
2	1	0	0	0	0	1	1
3	0	1	0	0	0	0	0
4	0	1	0	0	0	0	0
5	0	0	1	0	0	0	0
6	0	0	1	0	0	0	0

G



	0	1	2	3	4	5	6	7
0	0	1	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0
2	1	0	0	1	0	0	0	0
3	0	1	1	0	0	0	0	0
4	0	0	0	0	0	1	0	0
5	0	0	0	0	1	0	1	0
6	0	0	0	0	0	1	0	1
7	0	0	0	0	0	0	1	0

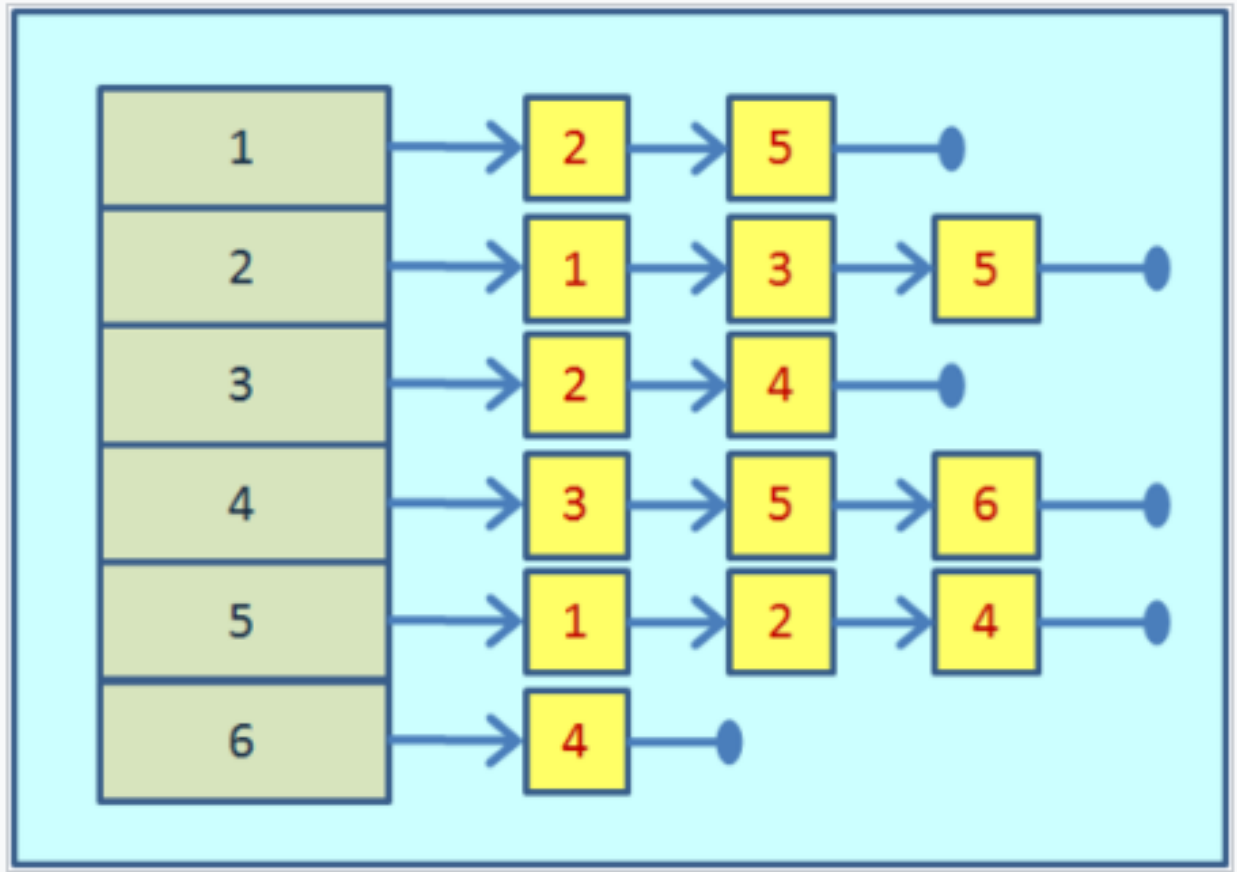
Representações de grafo - Listas de adjacências



Um grafo não dirigido com 6 vértices e 7 arestas.

O grafo da figura acima tem essa representação de lista de adjacência:

1	adjacente a	2,5
2	adjacente a	1,3,5
3	adjacente a	2,4
4	adjacente a	3,5,6
5	adjacente a	1,2,4
6	adjacente a	4



Lista de adjacências do grafo acima como encontrada em Cormen el al..

Códigos – Matriz e Listas de adjacências

- MatrizDeAdjacências.c

- ListaDeAdjacências.c
  - `void GRAPHremoveArc( Graph G, vertex v, vertex w); // implementar`
  - `void GRAPHshow( Graph G); // implementar`
  - `void GRAPHshowAll( Graph G); // implementar`

- Implementar o algoritmo de 8.2: “proc profr (v:nó)_”, versão recursiva.

Percursos em grafos

em profundidade (depth-first search - dfs)
- arestas que partem do vértice visitado por último

em largura (breadth-first search - bfs)
- arestas que partem do vértice visitado primeiro

guloso (greedy)
- arestas de menor custo (tipicamente procurando menor caminho)

Uma forma bem mais simples do procedimento para percurso em profundidade é a forma recursiva (como já foi visto antes, a utilização de pilhas é um modo de obter o efeito de chamadas recursivas):

```
proc profr (v: nó)_
    var w: nó;
    início
        execute visite (v);
        execute marque (v);
        para cada w adjacente a v faça
            se ¬ marcado (w) então execute profr (w)
    fim
```

Percursos em grafos

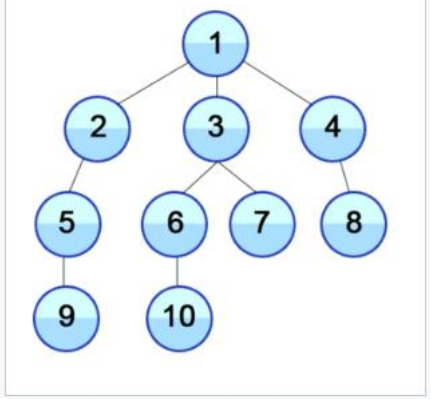
```
proc ampl (v: nó)_
    var t, w: nó;
    var f: fila;
    início
        execute visite (v);
        execute marque (v);
        execute enfileire (v, f);
        enquanto ¬ fila_vazia (f) faça
            início
                t ← frente (f);
                execute desenfileire (f);
                para cada w adjacente a t faça
                    se ¬ marcado (w) então
                        início
                            execute visite (w);
                            execute marque (w);
                            execute enfileire (w, f)
                        fim
            fim
    fim
```

```
proc prof (v: nó)_
    var t, w: nó;
    var p: pilha;
    início
        execute visite (v);
        execute marque (v);
        execute empilhe (v, p);
    enquanto ¬ pilha_vazia (p) faça
        início
            t ← topo (p);
            execute desempilhe (p);
            para cada w adjacente a t faça
                se ¬ marcado (w) então
                    início
                        execute visite (w);
                        execute marque (w);
                        execute empilhe (t, p);
                        t ← w
                    fim
        fim
fim
```

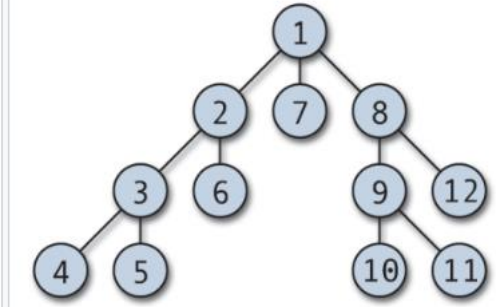
Universidade Federal de Santa Maria
1960

Percursos em grafos

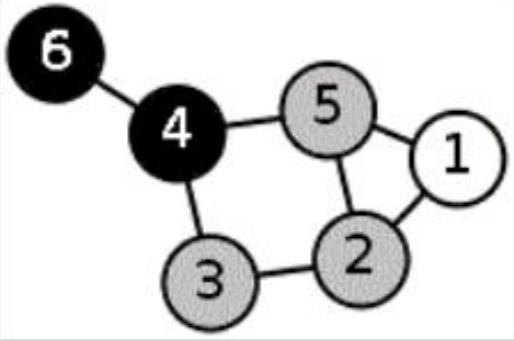
- Vídeo:
  - Algoritmos de busca em largura e profundidade em grafos.



Ordem dos vértices explorados na busca em largura



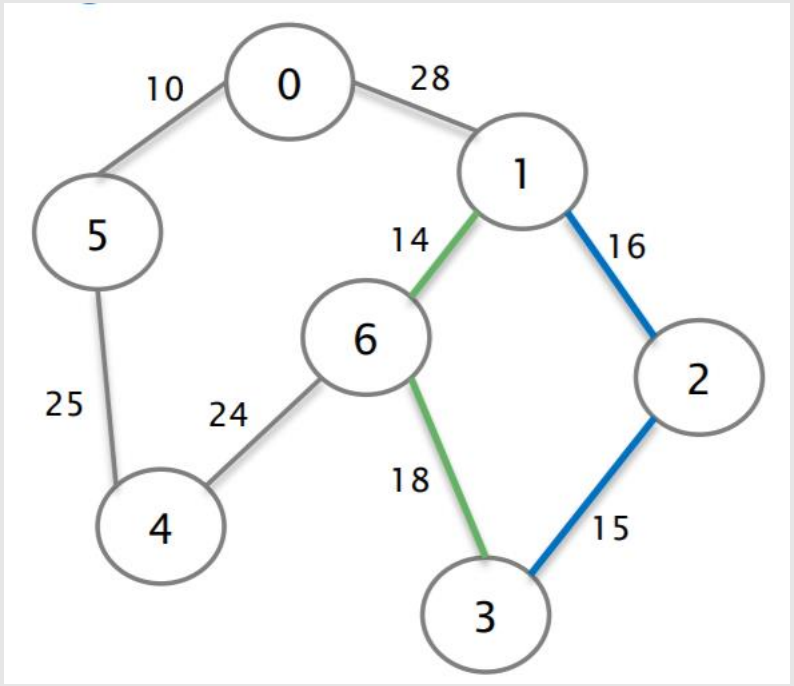
Ordem dos vértices explorados na busca em profundidade



Percurso realizado pelo algoritmo

Problemas comuns

- Árvores geradoras
- Caminho mais curto: caminho entre nós i e j com menor peso total de arcos



Algoritmo de Dijkstra

Entradas:

Um grafo ponderado G = (V,E,p)

Um vértice V do grafo

Saída:

Menor caminho entre V e
cada um dos nós do grafo

- roteamento em redes
- deslocamento de caminhões em trânsito pesado
- desenho de chips
- roteamento de mensagens em telecomunicações
- ...

- Vídeo.

Códigos – Percursos e Dijkstra

- BuscaEmProfundidade.c
- BuscaEmLargura.c
- ArvoreDeBuscaEmLargura.c
- Dijkstra.c

Atividades

- Revisar códigos em aberto
- Amanhã, 06/11, atividade em modo EAD.