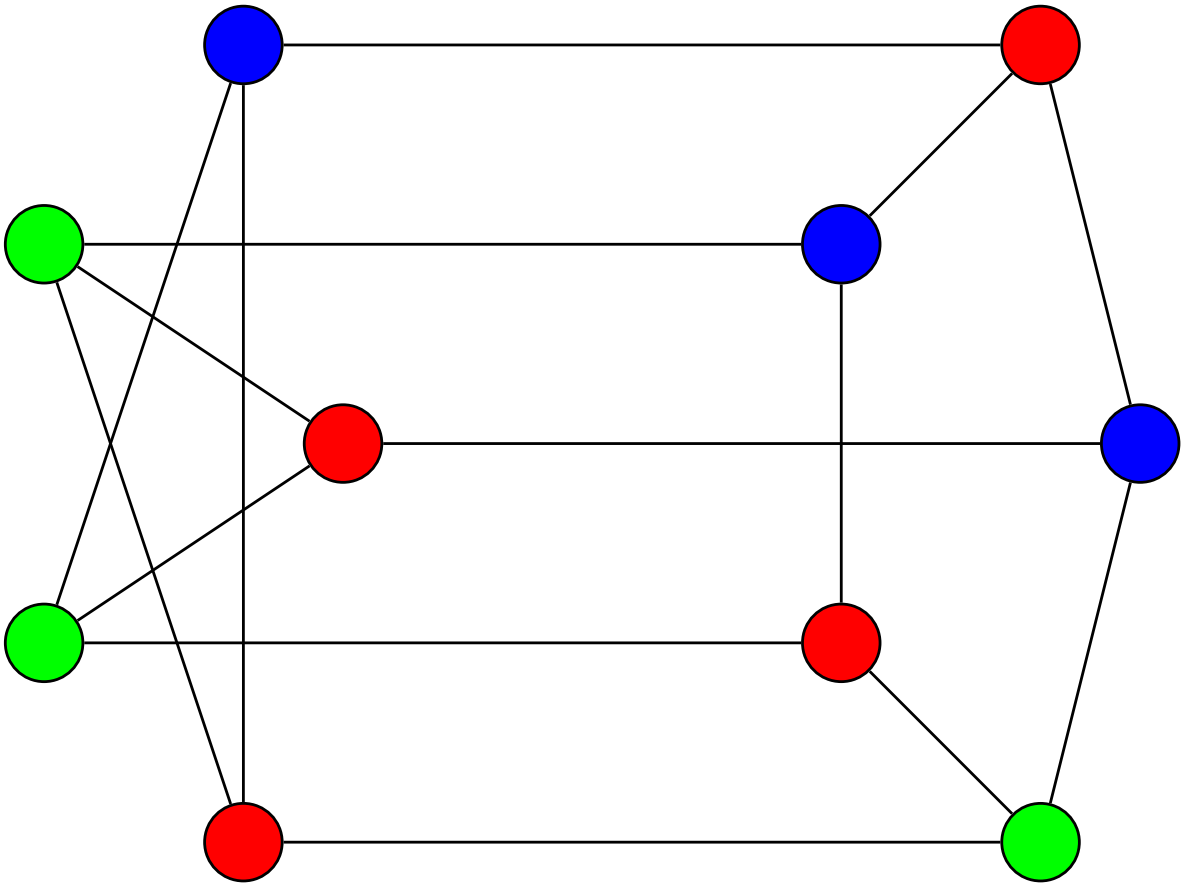
Centro de Matemática, Computação e Cognição
Universidade Federal do ABC
Profa. Carla Negri Lintzmayer

Notas de aula da disciplina Teoria dos Grafos

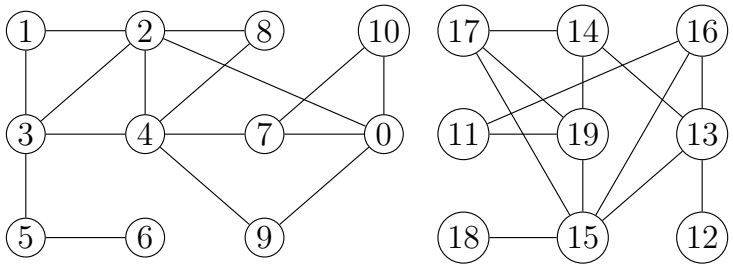


Esse material é um compilado dos seguintes:

- Sedgewick, R.. *Algorithms in C, part 5: graph algorithms*. 3rd ed. Addison-Wesley. 2002.
- Bondy, J. A.; Murty, U. S. R.. *Graph Theory*. Graduate Texts in Mathematics. Springer. New York. 2008.
- Lintzmayer, C. N.; Mota, G. O.. *Notas de aulas - Análise de algoritmos e estruturas de dados*. Em construção.

7 Aula 9: busca em profundidade (DFS - *Depth-First Search*)

- Expande a árvore pela vizinhança do vértice que foi visitado há menos tempo:
  - ordem "último a entrar, primeiro a sair";
  - os vértices são explorados de forma "agressiva".
- Esse algoritmo pode ser usado para:
  - encontrar componentes conexas;
  - encontrar caminhos entre vértices;
  - detectar ciclos;
  - verificar se um grafo é bipartido (e dar uma bipartição em caso positivo);
  - encontrar uma árvore geradora ou uma floresta geradora;
  - encontrar arestas e vértices de corte.
- Exemplo de execução:



- Por causa de algumas propriedades que veremos, é conveniente salvar a ordem na qual os vértices foram visitados.

```
1: Função ChamaDFS(G)
2:     sejam visitado, pred e ordem vetores indexados por vértices
3:     Para cada v ∈ V(G) faça
4:         ordem[v] ← 0
5:         visitado[v] ← 0
6:         pred[v] ← Null
7:     seja s ∈ V(G)
8:     tempo ← 0
9:     DFS(G, s)                                  ▷ DFS_pilha(G, s)
```

```
1:  Função DFS_PILHA(G, s)
2:      visitado[s] ← 1
3:      tempo ← tempo + 1
4:      ordem[s] ← tempo
5:      crie uma pilha vazia P
6:      EMPILHA(P, s)
7:      Enquanto P ≠ ∅ faça
8:          u ← TOPO(P)
9:          novos ← 0
10:         Se existe v ∈ N(u) com visitado[v] = 0 então        ▷ esconde um laço
11:             visitado[v] ← 1
12:             tempo ← tempo + 1
13:             ordem[v] ← tempo
14:             pred[v] ← u
15:             EMPILHA(P, v)
16:         Senão                          ▷ todos os vizinhos de u estão visitados
17:             u ← DESEMPILHA(P)
```

```
1:  Função DFS(G, s)
2:      visitado[s] ← 1
3:      tempo ← tempo + 1
4:      ordem[s] ← tempo
5:      Para cada v ∈ N(s) faça
6:          Se visitado[v] = 0 então
7:              pred[v] ← s
8:              DFS(G, v)
```

Lema 7.1. *Seja G um grafo e $s \in V(G)$. No fim da execução de DFS(G, s), um vértice v está visitado se e somente se existe sv-caminho em G.*

- Tempo de execução de DFS(G, s):
  - Note que são feitas $O(V)$ chamadas recursivas, pois só fazemos uma chamada para cada vértice ainda não visitado e um vértice não tem seu atributo visitado modificado.
  - Cada chamada recursiva faz algumas operações constantes sobre o vértice s e percorre sua vizinhança (laço **para** da linha 5).

  - Se utilizarmos matriz de adjacências, então o laço **para** leva tempo $\Theta(V)$ para executar uma única vez. Como ele executa no máximo uma vez para cada vértice, o tempo total é $\leq \sum_{s \in V(G)} \Theta(V) = O(V^2)$. Assim, o tempo total do algoritmo é $O(V) + O(V^2) = O(V^2)$.
  - Se utilizarmos listas de adjacências, então o laço **para** leva tempo $\Theta(d(u))$ para executar uma única vez. Como ele executa no máximo uma vez para cada vértice, o tempo total é $\leq \sum_{s \in V(G)} \Theta(d(u)) = O(E)$. Assim, o tempo total do algoritmo é $O(V) + O(E) = O(V + E)$.

- Portanto, a DFS é linear no tamanho da entrada.
- O Lema 7.1 mostra que DFS(G, s) visita todos os vértices que estão na mesma componente conexa de s.
- A árvore resultante da DFS é chamada *árvore de busca em profundidade* ou *árvore de DFS*.

Proposição 7.2. *Seja T uma árvore de DFS de G. Se $uv \in E(G)$, $uv \notin E(T)$ e $ordem[u] < ordem[v]$, então u é um ancestral de v.*

- Suponha que G é conexo e considere $uv \in E(G)$. Em uma busca, ou uv está na árvore ou é uma *aresta de retorno*. Podemos classificá-la ainda em um dos seguintes tipos durante a DFS, no momento em que estamos explorando os vizinhos de u:
  - *aresta da árvore*: se $visitado[v] = 0$;
  - *aresta para o pai*: se $pred[u] = v$;
  - *aresta de retorno para cima*: se $ordem[v] < ordem[u]$ e $pred[u] \neq v$;
  - *aresta de retorno para baixo*: se $ordem[v] > ordem[u]$.

Teorema 7.3. *Um grafo conexo G é acíclico se e somente se nenhuma aresta de retorno for encontrada por uma DFS sobre G.*

- Vamos usar a DFS para encontrar arestas de corte (que, claramente, não podem ser arestas de retorno).

Teorema 7.4. *Seja uv uma aresta de uma árvore de DFS, com $pred[v] = u$. Essa aresta é de corte se e somente se não existe uma aresta de retorno xy onde x é um descendente de v e y é um ancestral de u (note que é possível ter $x = v$ ou $y = u$).*

- Vamos manter qual é a menor ordem de um vértice que pode ser atingido por uma aresta de retorno com um extremo na subárvore enraizada em u em $low[u]$.
  - Assim, se $low[u] = ordem[u]$, então não há aresta de um descendente de u para um ancestral de u.

```
1:  Função ContaPontes(G)
2:      sejam visitado, pred, ordem, low vetores indexados por vértices
3:      Para todo v ∈ V(G) faça
4:          visitado[v] ← 0
5:          pred[v] ← Null
6:          ordem[v] ← 0
7:          low[v] ← 0
8:      seja s ∈ V(G) qualquer
9:      tempo ← 0
10:     qtd ← 0
11:     Ponte(G, s)
12:     Devolve qtd
```

```
1:  Função Ponte(G, s)
2:      visitado[s] ← 1
3:      tempo ← tempo + 1
4:      ordem[s] ← tempo
5:      low[s] ← tempo
6:      Para v ∈ N[s] faça
7:          Se visitado[v] = 0 então
8:              pred[v] = s
9:              Ponte(G, v)
10:             Se low[s] > low[v] então
11:                 low[s] ← low[v]
12:             Se low[v] = ordem[v] então
13:                 qtd ← qtd + 1          ▷ A aresta sv é aresta de corte
14:         Senão Se v ≠ pred[s] então
15:             Se low[s] > ordem[v] então
16:                 low[s] ← ordem[v]
```

Lema 7.5. *Seja G um grafo e $s \in V(G)$ um vértice qualquer. O algoritmo Ponte(G,s) corretamente detecta quantas arestas de corte existem na componente conexa de s.*