CT-234

Estruturas de Dados, Análise de Algoritmos e Complexidade Estrutural

Carlos Alberto Alonso Sanches

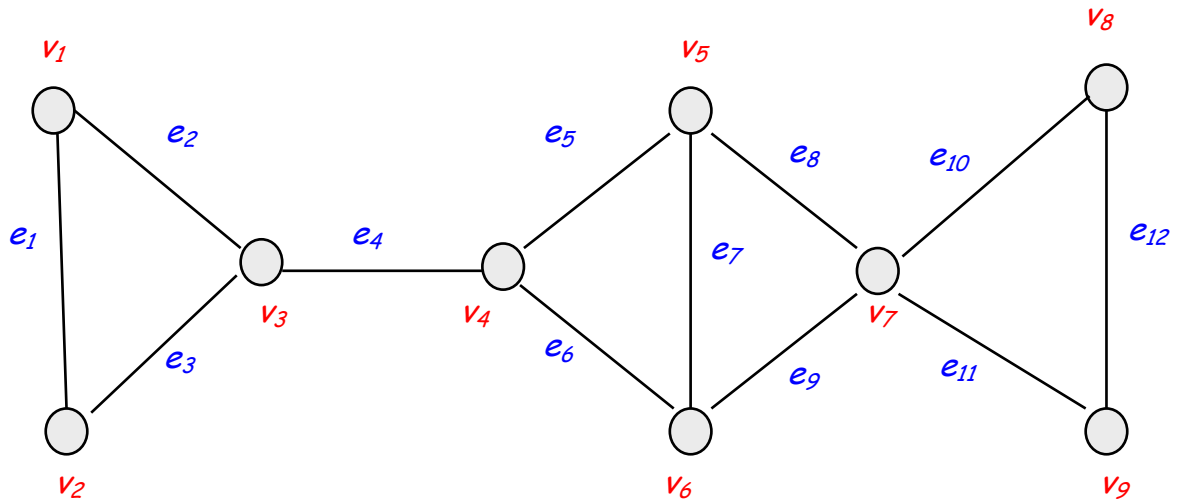
CT-234

8) Algoritmos em grafos

Conceitos básicos, representações, explorações sistemáticas

Definição

- Um grafo G=(V,E) é formado pelos vértices $V = \{v_1, v_2, \ldots, v_n\}$ e pelas arestas $E = \{e_1, e_2, \ldots, e_m\}$.
- Consideraremos sempre que $|V| = n$ e $|E| = m$.
- Um exemplo:



$V = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9\}$ $n = 9$

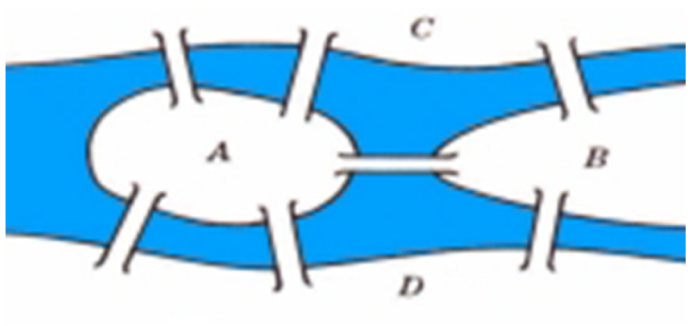
$E = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8, e_9, e_{10}, e_{11}, e_{12}\}$ $m = 12$

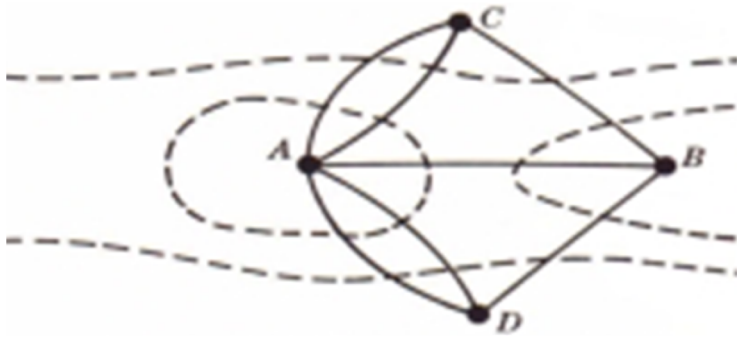
Arestas e vértices

- Uma aresta $e \in E$ é um par não-ordenado (u,v), onde $u,v \in V$.
- Neste caso, dizemos que os vértices u e v são adjacentes entre si, e que a aresta e é incidente em u e em v.
- Uma aresta $e = (u,v)$ é chamada de laço quando $u = v$.
- $d(u)$ é o grau do vértice u, isto é, o número de incidências em u.
- É fácil observar que $\Sigma_{u \in V} d(u) = 2m$

Origem histórica

- Na cidade de Königsberg (atual Kaliningrado), havia sete pontes sobre o rio Pregel:

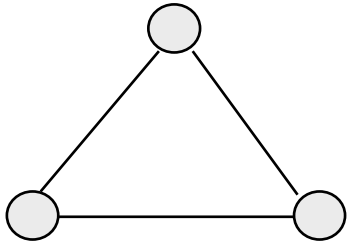




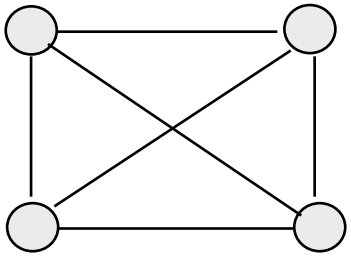
- Será possível fazer um passeio pela cidade, começando e terminando no mesmo local, e passando uma única vez em cada ponte?
- Euler (1736) afirmou que um grafo conexo tem esse passeio se e somente se cada um dos seus vértices tem grau par.
- Todo grafo com essa propriedade é chamado de *euleriano*.

Grafos regulares e completos

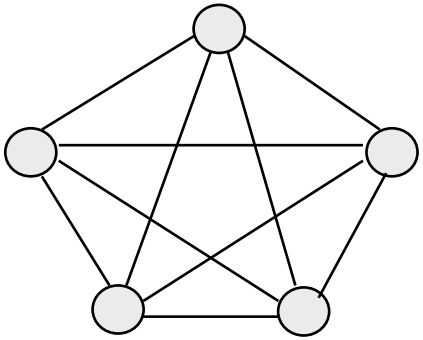
- Grafo k-regular, com $k \geq 0$: $u \in V \Leftrightarrow d(u) = k$
- Grafo completo K_n: $(u,v \in V, u \neq v) \Leftrightarrow (u,v) \in E$
- Exemplos:



K_3



K_4



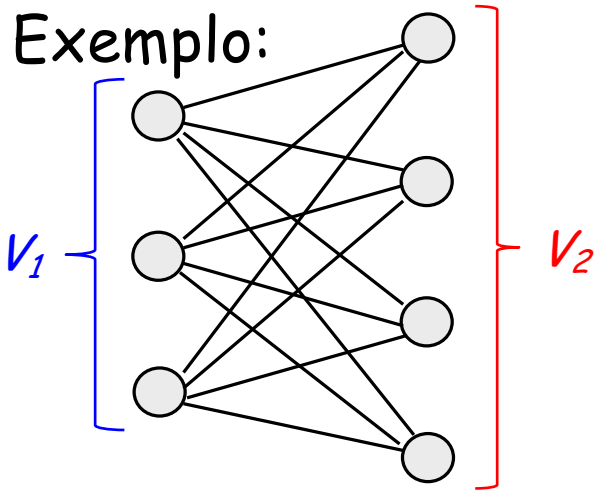
K_5

- É fácil verificar que K_n:
  - É (n-1)-regular.
  - Tem n(n-1)/2 arestas.

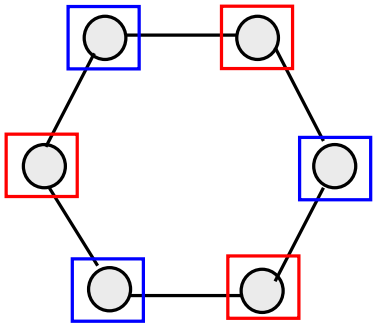
Grafos bipartidos ou bicoloridos

- Um grafo G é bipartido (ou bicolorido) quando seus vértices podem ser particionados em dois subconjuntos V_1 e V_2 tais que qualquer aresta de G possui uma extremidade em V_1 e outra em V_2.

- Exemplo:

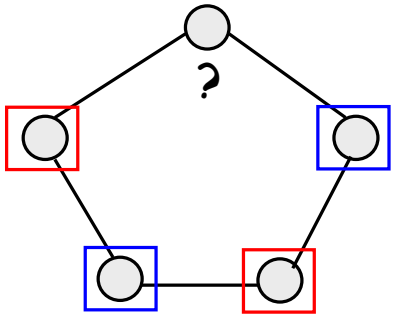


$K_{a,b}$: grafo bipartido completo onde $|V_1| = a$ e $|V_2| = b$



É bipartido?

SIM

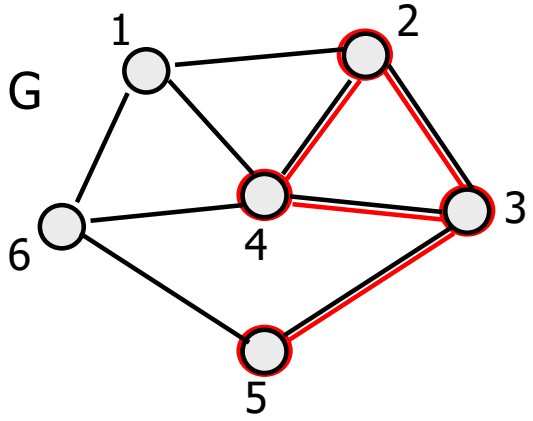


É bipartido?

NÃO

Subgrafos

- O grafo $G'=(V',E')$ é um subgrafo de $G=(V,E)$ se $V' \subseteq V$ e $E' \subseteq E$, e todas arestas de E' têm seus vértices em V'.
- Quando $V'=V$, G' é chamado de subgrafo gerador de G.
- Seja $X \subseteq V$ e $E(X)$ o subconjunto das arestas de E com ambos os vértices em X. Dizemos que $G(X)=(X,E(X))$ é o subgrafo de G induzido por X.
- Exemplo:

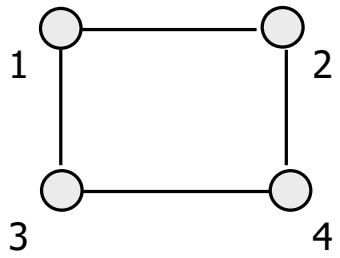


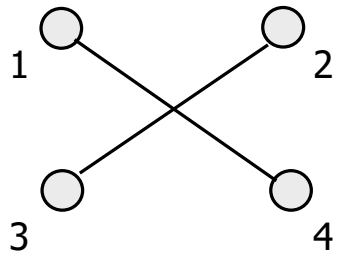
$X = \{2, 3, 4, 5\}$

$G(X)$

Grafos complementares

- Dado um grafo, seu complementar possui os mesmos vértices, mas tem somente as arestas que faltam no original.
- Formalmente, dado um grafo $G=(V,E)$, seu complementar será $G'=(V,E')$ tal que:
  - $(u,v) \in E \Rightarrow (u,v) \notin E'$
  - $(u,v) \notin E \Rightarrow (u,v) \in E'$
- Exemplo:

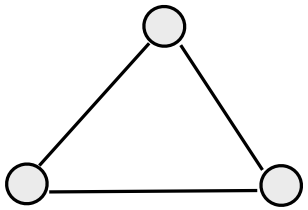




Grafos planares

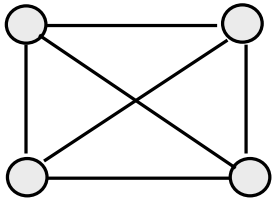
- Um grafo é planar se ele pode ser representado no plano de tal modo que não haja interseções entre suas arestas.
- Exemplos:

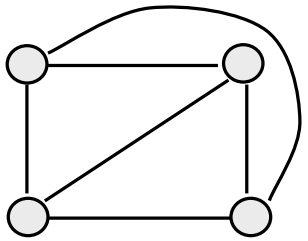
K_3 ?



É planar

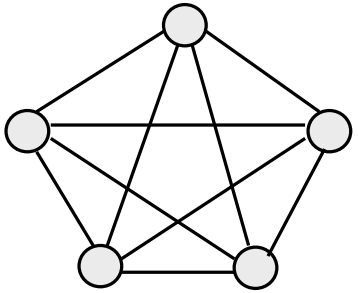
K_4 ?





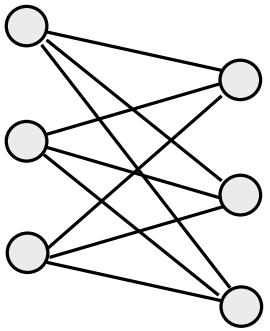
É planar

K_5 ?



Não é planar

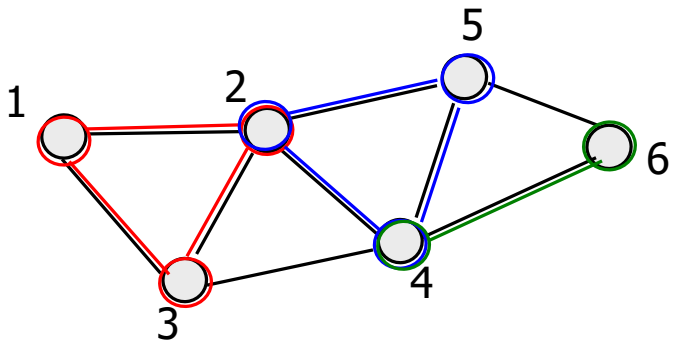
$K_{3,3}$?



Não é planar

Cliques

- Clique é um subconjunto de V que induz um grafo completo.
- Exemplos:



$C_1 = \{1, 2, 3\}$

$C_2 = \{2, 4, 5\}$

$C_3 = \{4, 6\}$

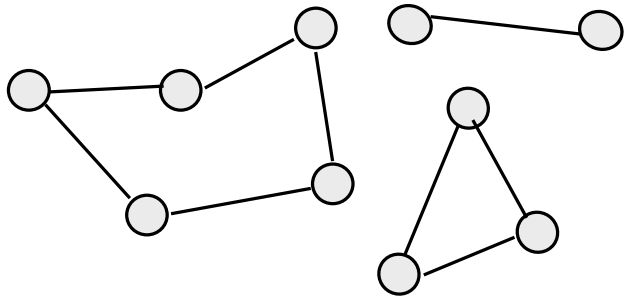
$C_4 = \{3\}$

Sequências de vértices

- Caminho é uma sequência alternada de vértices e arestas, onde cada aresta é incidente tanto ao vértice que a antecede como ao que a segue.
- Caminho simples é um caminho no qual cada vértice aparece uma única vez.
- Comprimento de um caminho é o seu número de arestas.
- Ciclo ou circuito é um caminho que começa e termina no mesmo vértice.

Componentes conexas

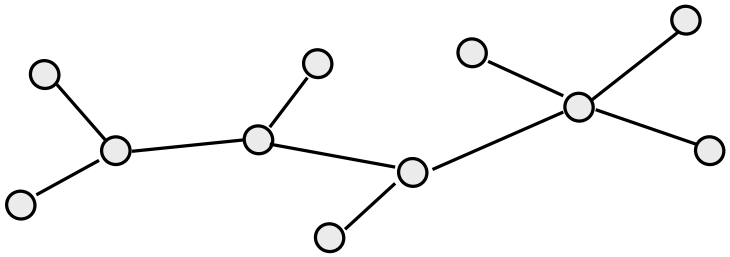
- Dois vértices *v* e *u* são conectados se houver um caminho entre eles. Componentes conexas são subconjuntos maximais de vértices conectados entre si.
- Um grafo é conexo se tiver uma única componente conexa, ou seja, se todos seus vértices estiverem conectados entre si.
- Exemplo:



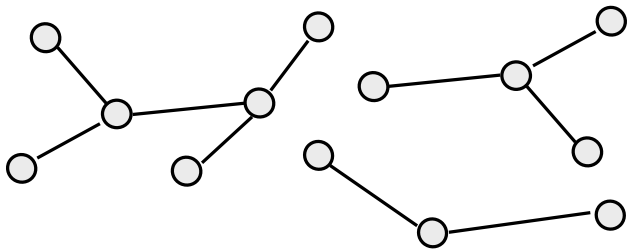
Grafo com 3 componentes conexas

Árvores e florestas

- Árvore é um grafo conexo sem circuitos.
- Exemplo:

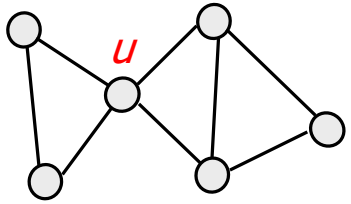


- Portanto, todo caminho simples é uma árvore.
- Floresta é um grafo cujas componentes conexas são árvores.
- Exemplo:

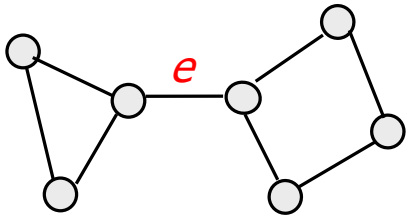


Vértices e arestas de corte

- Em um grafo G, $u \in V$ é chamado de vértice de corte (ou ponto de articulação) se a sua remoção desconecta G.
- Exemplo:

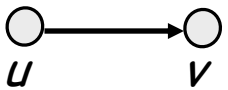


- Se uma componente conexa de um grafo não possui vértices de corte, ela é chamada de componente biconexa.
- Analogamente, uma aresta e, cuja remoção ocasiona a desconexão do grafo, recebe o nome de ponte (ou aresta de corte).
- Exemplo:



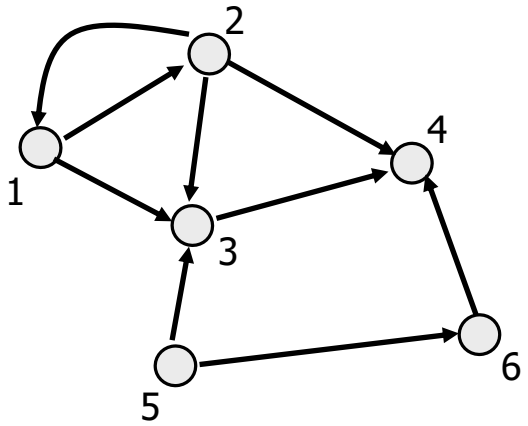
Digrafos ou grafos orientados

- Digrafos são grafos orientados, isto é, suas arestas possuem direção e são chamadas de arcos.
- Em um digrafo $G=(V,E)$, um arco $e \in E$ é um par ordenado (u,v), onde $u,v \in V$.



- v é sucessor de u
- u é predecessor de v

- Cada vértice v tem um grau de saída $d^+(v)$ e um grau de entrada $d^-(v)$, que correspondem respectivamente ao total de arcos que saem ou chegam em v.
- Exemplo:



$d^+(4) = 0$

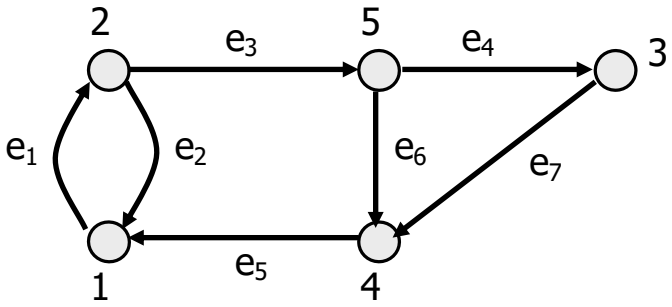
$d^-(4) = 3$

$d^+(6) = 1$

$d^-(1) = 1$

Sequências de arcos

- Caminho é uma sequência de arcos $e_1, e_2, \ldots, e_q$ tal que a extremidade inicial de e_i coincide com a final de e_{i-1}, $1 < i \leq q$.
- Ciclo ou circuito é um caminho que começa e termina no mesmo vértice.
- Exemplo:



e_3, e_4, e_7, e_5: caminho entre os vértices 2 e 1

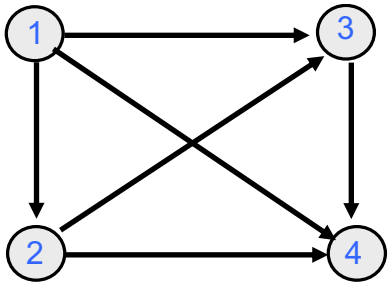
e_3, e_6, e_5, e_1: ciclo ou circuito

Ordenação topológica

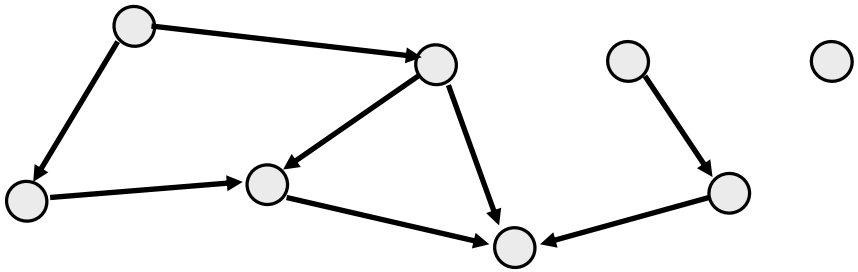
- Uma ordenação topológica de um digrafo $G=(V,E)$ corresponde a uma bijeção $f: V \rightarrow \{1,2, \ldots, n\}$ tal que, para todo arco $(u,v) \in E$, $f(u) < f(v)$.
- Em outras palavras, deseja-se numerar os vértices de tal modo que, se houver em G um caminho de u até v, então o número de u será menor que o de v.
- É possível provar que um digrafo G admite uma ordenação topológica se e somente se for acíclico.
- Se os vértices forem alinhados de acordo com uma ordenação topológica, todos os arcos terão uma mesma direção.

Exercício

- Encontre uma ordenação topológica para os digrafos abaixo.
  - Dica: utilize os graus de entrada dos vértices.



Uma única ordenação



Mais de uma ordenação

Uma solução

- Calcular o grau de entrada de todos os vértices.
- Começar com o conjunto de vértices que têm grau de entrada nulo.
- Para cada um desses vértices, dar um numeração baixa, eliminá-los do conjunto e descontá-los nos graus de entrada dos seus sucessores. Se algum desses sucessores passar a ter grau de entrada nulo, incluí-lo no conjunto.
- A execução termina quando esse conjunto se torna vazio.

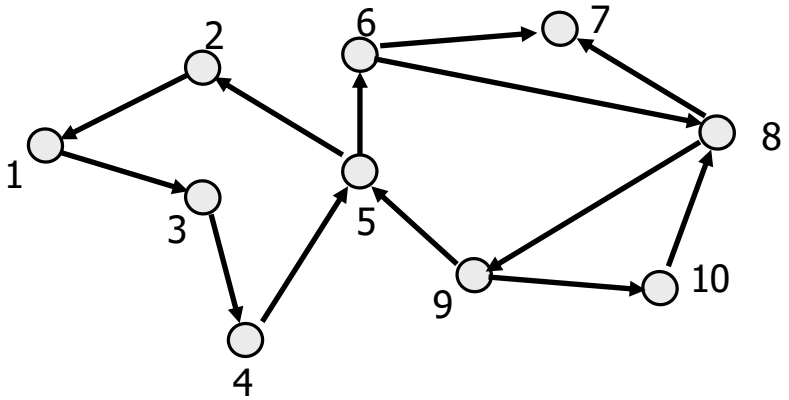
Algoritmo

```
TopSort() {
    counter = 0;
    for v ∈ V
        calcular indegree(v);
    queue q;
    for v ∈ V
        if (indegree(v) == 0) q.enqueue(v);
    while (!q.isEmpty()) {
        v = q.dequeue();
        f[v] = ++counter;
        for <v,w> ∈ E
            if(--indegree(w) == 0) q.enqueue(w);
    }
    if (counter != n) print("Grafo é cíclico");
}
```

- Ao invés de fila, poderia ser utilizada uma pilha?

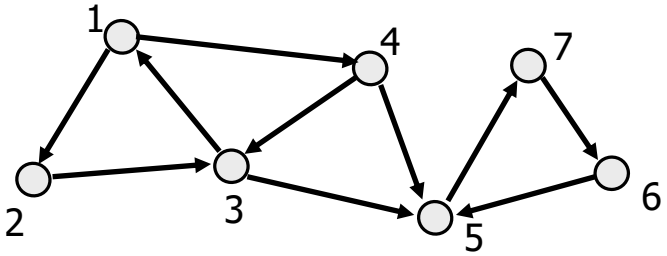
Componentes fortemente conexas

- Em um digrafo, dois vértices v_i e v_j estão conectados entre si se existem caminhos de v_i a v_j e de v_j a v_i. Desse modo, surge analogamente o conceito de componente fortemente conexa (CFC).
- Exemplos:



{1, 2, 3, 4, 5, 6, 8, 9, 10}

{7}



{1, 2, 3, 4}

{5, 6, 7}

- É possível criar um digrafo acíclico (DAG) considerando cada CFC como um vértice e mantendo os arcos incidentes em CFCs distintas.

Modelagem de problemas com grafos

- Problema: Em um vídeo-game, deseja-se um projeto de rotas para que personagens possam circular através de salas com objetos.
- Solução: Gerar um grafo onde cada vértice seria um lugar válido, enquanto as arestas representariam vizinhança. Neste caso, se poderia utilizar um algoritmo de caminho mínimo.

Modelagem de problemas com grafos

- Problema: No sequenciamento de DNA, são obtidos experimentalmente pequenos fragmentos. Para cada fragmento, existem alguns que necessariamente devem estar à sua esquerda, outros à sua direita, e outros sem qualquer restrição. Como determinar um sequenciamento consistente?
- Solução: Criar um digrafo onde cada vértice representaria um fragmento. As arestas (l,f) e (f,r) indicariam que os fragmentos l e r devem estar, respectivamente, à esquerda e à direita de f. Em seguida, basta encontrar uma ordenação topológica desse digrafo.

Modelagem de problemas com grafos

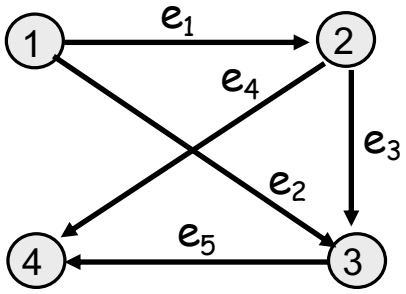
- Problema: Ao transferir arquivos de um sistema UNIX para outro DOS, é preciso reduzir o tamanho dos nomes de centenas de arquivos, pois devem ter no máximo 8 caracteres. Como garantir que não haja conflitos?
- Solução: Construir um grafo onde os nomes sejam vértices, e as arestas unem nomes originais com suas possíveis reduções. O problema agora é determinar um conjunto independente de arestas em um grafo bipartido.

Representações de grafos

- Há algumas estruturas de dados apropriadas para representar grafos.
- Principais representações:
  - Matriz de incidências
  - Matriz de adjacências
  - Lista de adjacências
    - Com ponteiros
    - Com vetores
  - Lista de arcos

Matriz de incidências

- Matriz de incidências é formada por n linhas (uma para cada vértice) e m colunas (uma para cada aresta ou arco).
- A posição a_{ij} dessa matriz indica se a aresta ou o arco e_j incide sobre o vértice v_i.
- Exemplo:



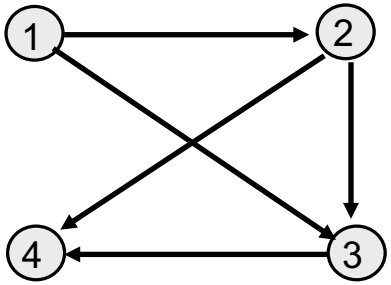
$$A_{n\times m} = \begin{bmatrix} +1 & +1 & 0 & 0 & 0 \\ -1 & 0 & +1 & +1 & 0 \\ 0 & -1 & -1 & 0 & +1 \\ 0 & 0 & 0 & -1 & -1 \end{bmatrix}$$

Grafo não dirigido: haveria somente valores 1 na matriz

Tamanho da estrutura: $\Theta(n.m)$

Matriz de adjacências

- Matriz de adjacências é formada por n linhas e n colunas.
- A posição a_{ij} dessa matriz indica se o vértice v_j é sucessor ou não do vértice v_i.
- Exemplo:



$$A_{n \times n} = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

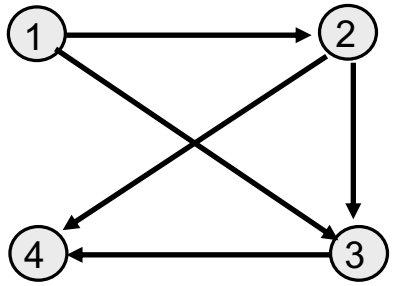
Grafo não dirigido: a matriz seria simétrica

Tamanho da estrutura: $\Theta(n^2)$

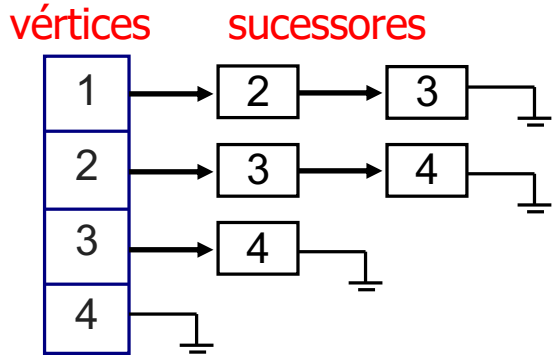
Útil quando grafo é denso: $m \sim n^2$

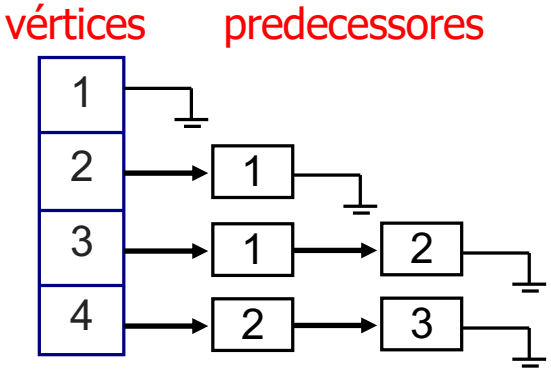
Lista de adjacências

- Lista de adjacências é formada por um vetor de *n* ponteiros, onde cada vértice aponta para seus sucessores ou predecessores.
- Exemplo:



Grafo não dirigido: haveria o dobro de nós



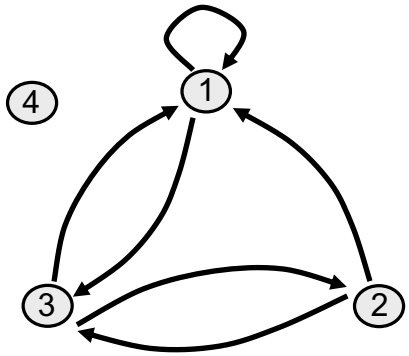


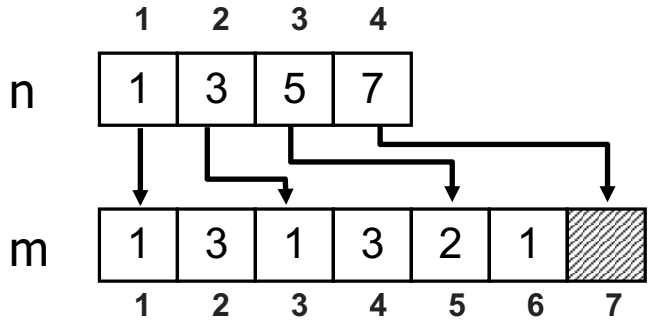
Tamanho da estrutura: $\Theta(n+m)$

Útil quando grafo é esparso: $m << n^2$

Representações alternativas

- Lista de adjacências através de vetores

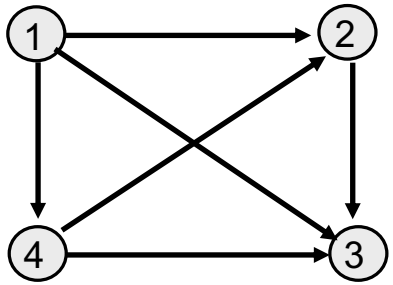




Grafo não dirigido:
o segundo vetor teria
o dobro do tamanho

Tamanho da estrutura: Θ(n+m)

- Lista de arcos



S(.)	1	1	1	2	4	4
T(.)	2	3	4	3	2	3

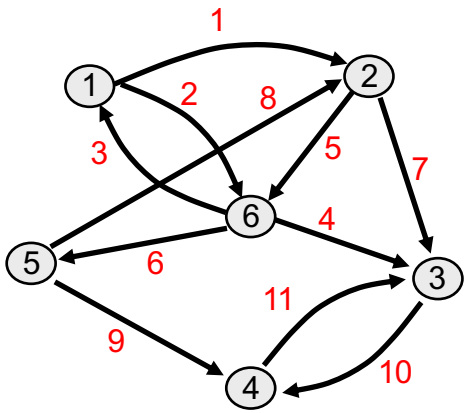
Grafo não dirigido:
mesma estrutura

Tamanho da estrutura: Θ(m)

Representação mais adequada

- Alguns critérios para se escolher a melhor representação:
  - Espaço de armazenamento (depende do tamanho do grafo)
  - Teste de pertinência de uma aresta (matriz)
  - Verificação do grau de um vértice (lista)
  - Inserção ou remoção de uma aresta (matriz)
  - Percurso no grafo (lista)
- Geralmente, a lista de adjacências costuma ser mais vantajosa.

Outro exemplo



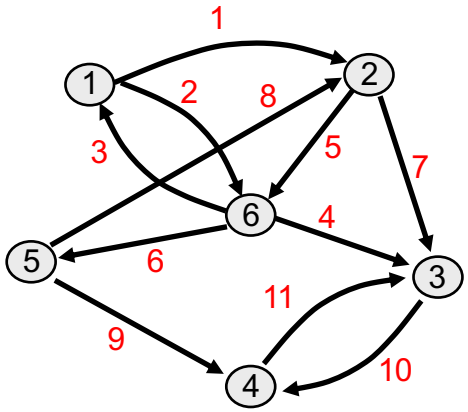
- Matriz de adjacências

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \end{bmatrix}$$

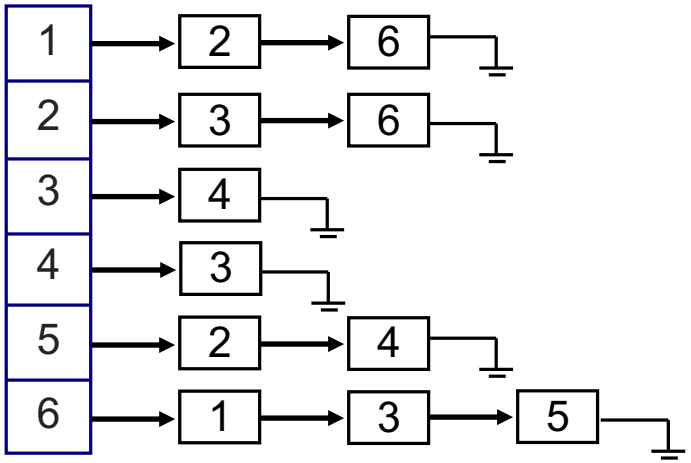
- Matriz de incidências

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{cccccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \end{array} \\ \begin{bmatrix} +1 & +1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & +1 & 0 & +1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & -1 & 0 & 0 & +1 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & +1 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & +1 & +1 & 0 & 0 \\ 0 & -1 & +1 & +1 & -1 & +1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

Outro exemplo



- Lista de adjacências



- Lista de arcos

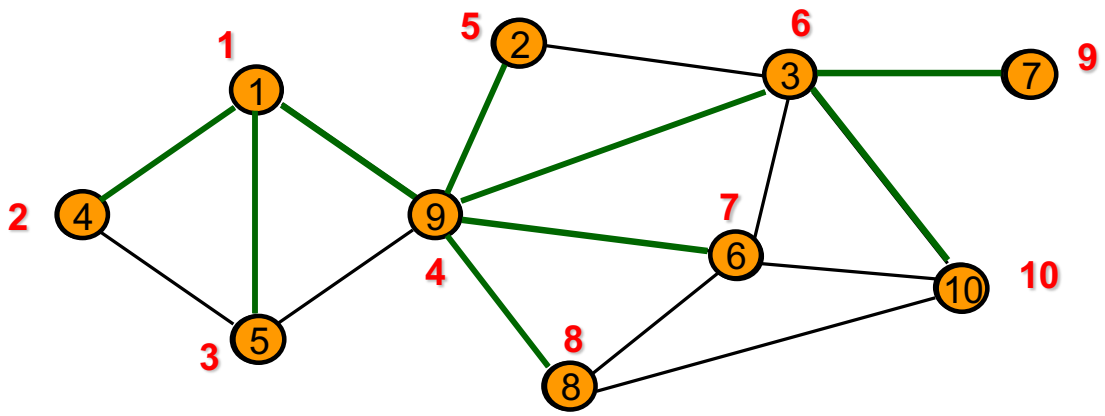
1	2	3	4	5	6	7	8	9	10	11
1	1	6	6	2	6	2	5	5	3	4
2	6	1	3	6	5	3	2	4	4	3

Exploração sistemática de um grafo

- Explorar um grafo é percorrê-lo completamente, visitando todos os vértices e as arestas.
- A ordem dessas visitas depende:
  - do vértice onde a exploração começa;
  - da ordem de armazenamento dos vértices e das arestas na estrutura de dados;
  - do tipo de exploração: em largura ou em profundidade.

Em largura (*breadth-first search*)

- Tática: enquanto for possível, examinar todos os vértices à mesma distância do vértice corrente; quando não for mais possível, aprofundar.
- Exemplo (supomos armazenamento em ordem crescente):



- Uma aplicação: a exploração em largura permite encontrar, por exemplo, as distâncias e os menores caminhos entre os vértices.

Exploração em largura

```
BFS(s) {
    desmarcar todos os vértices;
    int cont = 0;
    queue q;
    marcar s;
    expl[s] = ++cont;
    enqueue(q,s);
    while (!isEmpty(q)) {
          curr = dequeue(q);
          // explorando curr
          for <curr,v> ∈ E {
              if v está desmarcado {
                 marcar v;
                 expl[v] = ++cont;
                 enqueue(q,v);
              }
          }
    }
}
```

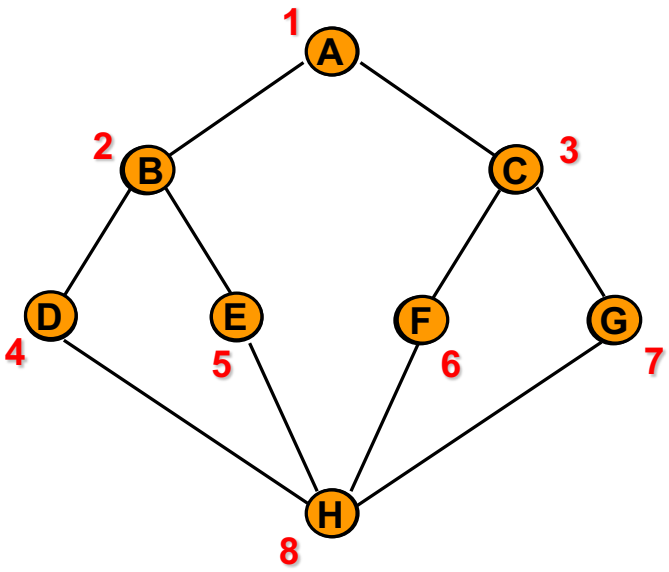
Código aplicável a grafos não orientados e conexos

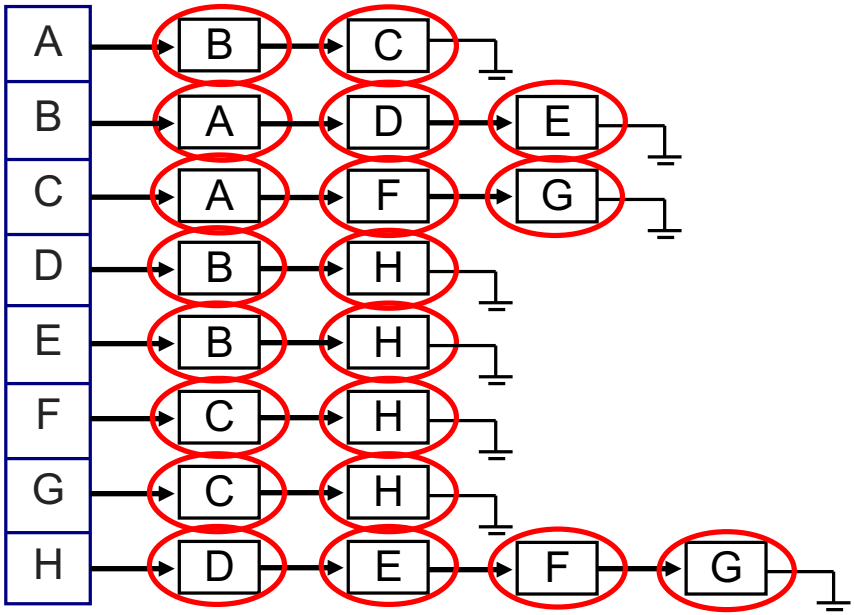
Cada vértice recebe um número de exploração (equivalente a marcá-lo)

Exploração em largura

- Durante a execução deste algoritmo, cada vértice pode ficar em três estados:
  - desmarcado (portanto, fora da fila): ainda não foi atingido;
  - marcado e na fila: atingido, mas não completamente explorado;
  - marcado e fora da fila: já explorado.
- Cada vértice entra na fila uma única vez, e cada aresta é visitada duas vezes. Portanto, sua complexidade de tempo é $\Theta(n+m)$.

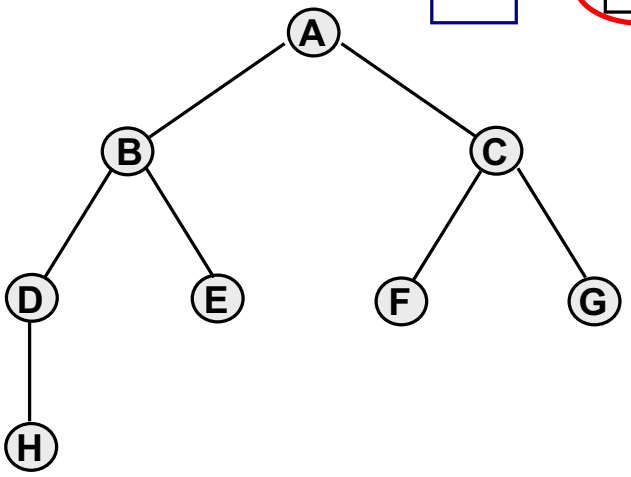
Exemplo





não visitado

visitado

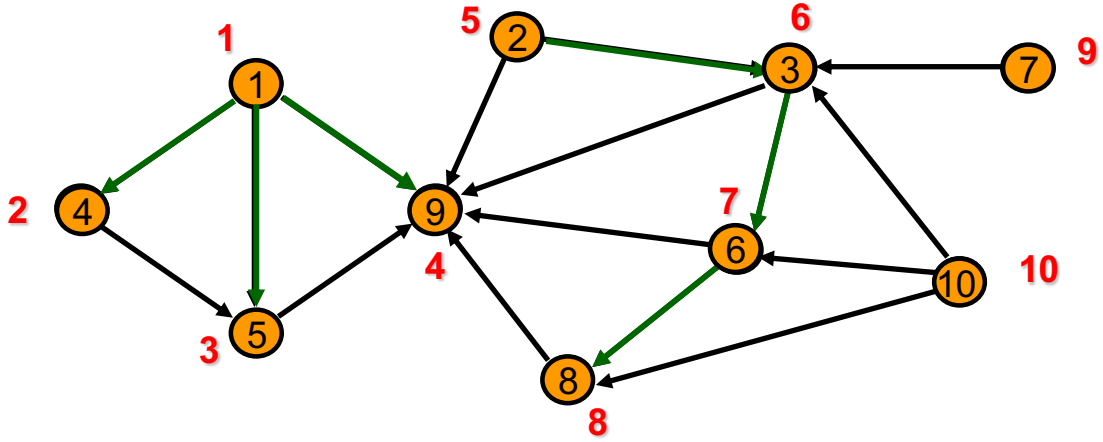


Árvore de exploração em largura

Exploração em largura (digrafos)

```
int cont;
queue q;

TravessiaBFS(s) {
    desmarcar todos os vértices;
    cont = 0;
    BFS(s);
    for v ∈ V {
        if v está desmarcado
            BFS(v);
    }
}
```

Código adicional



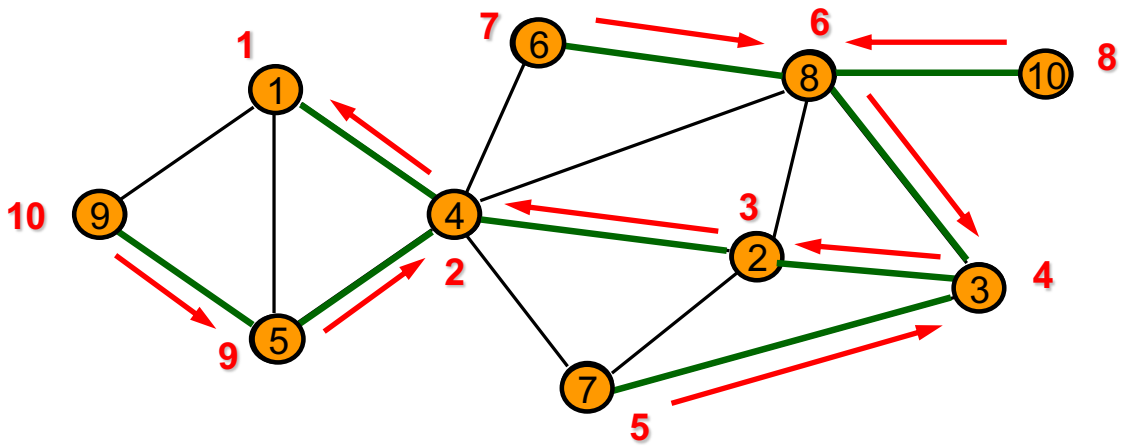

```
BFS(v) {
    marcar v;
    expl[v] = ++cont;
    enqueue(q,v);
    while (!isEmpty(q)) {
        curr = dequeue(q);
        // explorando curr
        for <curr,u> ∈ E {
            if u está desmarcado {
                marcar u;
                expl[u] = ++cont;
                enqueue(q,u);
            }
        }
    }
}
```

Tempo: $\Theta(n+m)$

Solução análoga para grafos desconexos

Em profundidade (*depth-first search*)

- Tática: enquanto for possível, aprofundar-se no grafo; quando não for mais possível, recuar um nível.
- Exemplo (supomos armazenamento em ordem crescente):



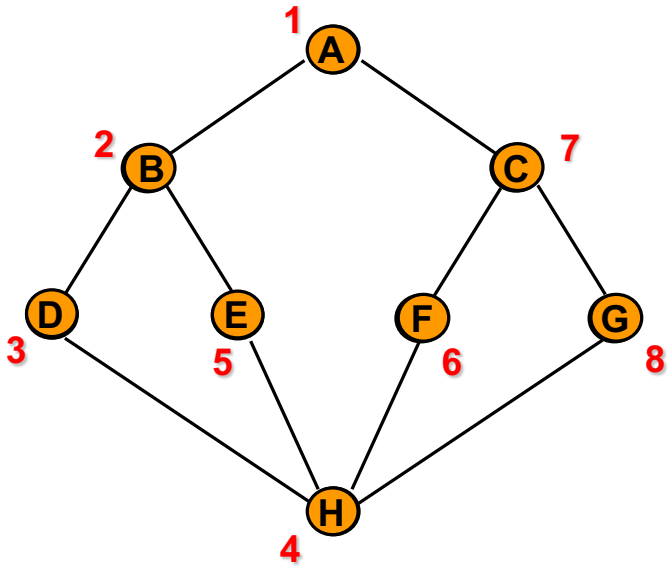
- A exploração em profundidade possibilita respostas a várias questões. Por exemplo: ordenação topológica; se o grafo é acíclico ou conexo; bicoloração dos vértices; quais são as suas componentes conexas e os eventuais vértices e arestas de corte.

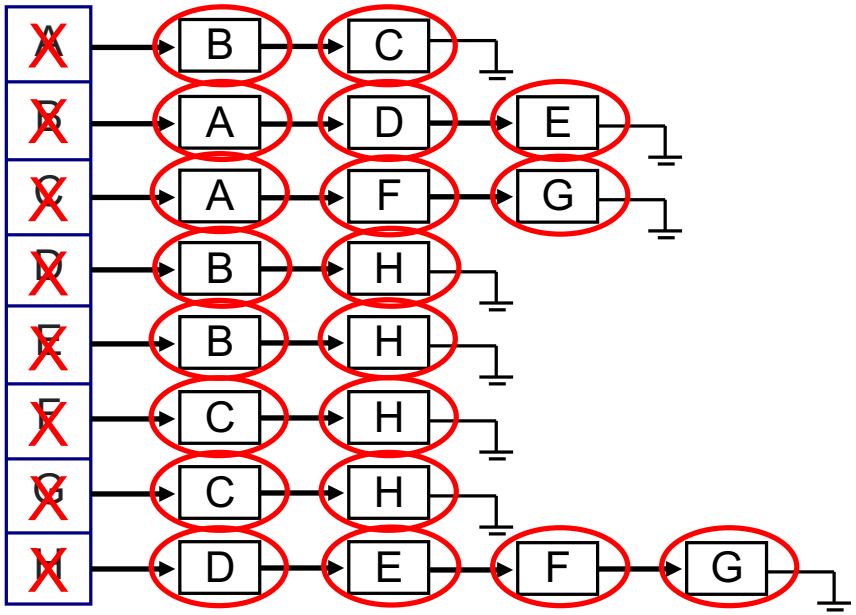
Versão recursiva

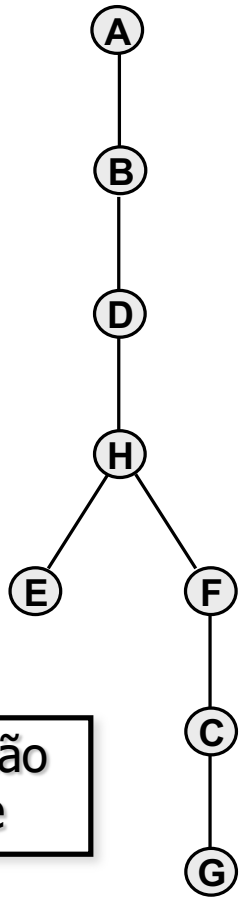
```
int cont = 0;
desmarcar todos os vértices;

DFS(s) {
    marcar s;
    expl[s] = ++cont;
    // explorando s
    for <s,v> ∈ E {
        if v está desmarcado
            DFS(v);
    }
}
```

- Analogamente à exploração em largura, a complexidade de tempo também é $\Theta(n+m)$.

Exemplo







não visitado

visitado

Árvore de exploração em profundidade

Versão iterativa

```
DFS(s) {
    desmarcar todos os vértices;
    stack P;
    int cont = 0;
    marcar s;
    push(P,s);
    while (!isEmpty(P)) {
          curr = top(P);
          pop(P);
          expl[curr] = ++cont;
          // explorando curr
          for <curr,v> ∈ E {
              if v está desmarcado {
                 marcar v;
                 push(P,v);
              }
          }
    }
}
```

Pequena diferença em relação à versão recursiva: inicialmente, marca e empilha os vértices vizinhos; depois, numera-os à medida que são desempilhados

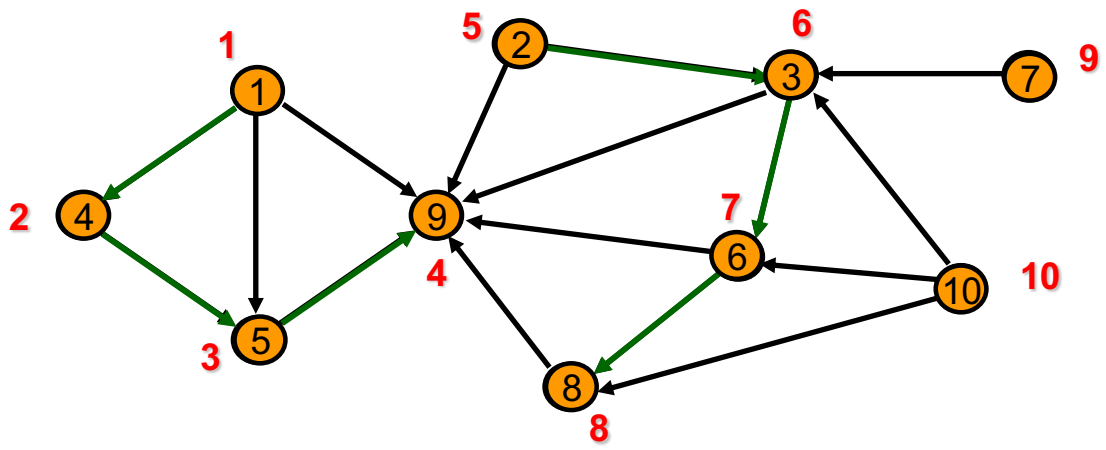
A ordem de empilhamento é diferente da versão recursiva: gerará outra árvore de exploração!

Exploração em profundidade (digrafos)

```
int cont;

TravessiaDFS(s) {
    desmarcar todos os vértices;
    cont = 0;
    DFS(s);
    for v ∈ V {
        if v está desmarcado
            DFS(v);
    }
}
```

Código adicional

```
DFS(v) {
    marcar v;
    expl[v] = ++cont;
    // explorando v
    for <v,u> ∈ E {
        if u está desmarcado
            DFS(u);
    }
}
```



Tempo: $\Theta(n+m)$

Solução análoga para grafos desconexos

Exercícios

- Descreva um algoritmo que, em tempo $O(n+m)$, identifique se um grafo é cíclico ou não.
  - Durante a exploração em profundidade, se em algum momento se atinge um vértice marcado mas ainda em exploração, então existe pelo menos um ciclo.
  - Esse ciclo pode ser obtido com o uso de uma pilha explícita.
- Como aproveitar essa exploração em profundidade para encontrar uma ordenação topológica?
  - Basta numerar também o término das explorações, e utilizar essa numeração em ordem inversa.