

# Teoria dos Grafos

## Aula 3

### Aula passada

- Definições
- Representando grafos
- Matriz e lista

### Aula de hoje

- Explorando grafos
- Mecanismos genéricos
- Busca em Largura-BFS
- Busca em Profundidade-DFS
- Complexidade
- Conectividade

# Busca em Grafos

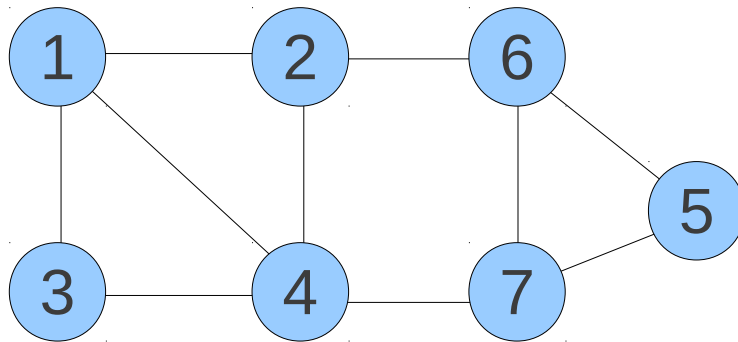
- Problema fundamental em grafos

**Como *explorar* um grafo de forma sistemática?**

- Muitas aplicações são abstraídas como problemas de busca
- Muitos algoritmos utilizam fundamentos similares

# Busca em Grafos

- Como saber se existe caminho entre dois vértices?
  - de maneira eficiente



- **Idéia:** evitar explorar vértices já explorados
  - marcar os vértices!
  - vértice: *descoberto* ou *explorado*

# Busca em Grafos

- Definir vértice inicial (origem ou raiz)
- Explorar e marcar vértices
  - *Descoberto*: vértice foi descoberto (visitado pela primeira vez)
  - *Explorado*: todas as arestas incidentes ao vértice foram exploradas e vizinhos descobertos
- Algoritmo genérico?

# Algoritmo Genérico

- Passo inicial
  - desmarcar todos os vértices
  - selecionar origem e marcá-lo *descoberto*
- Passo geral (enquanto houver vértice descoberto)
  - Selecionar vértice descoberto,  $u$
  - Considerar aresta não explorada,  $(u, v)$
  - Se  $v$  não estiver marcado, marcar  $v$  como *descoberto*
  - Marcar  $u$  *explorado* quando não houver mais arestas incidentes a  $u$  a serem exploradas

# Ordenação da Exploração

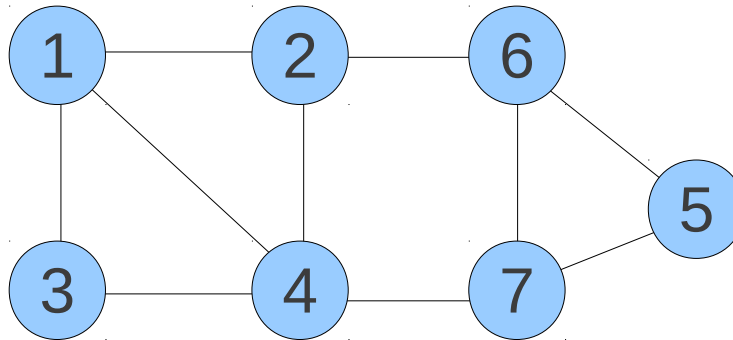
- Ordem de visita dos vértices e arestas?

**Algoritmo genérico não estabelece ordem**

- Qual é uma possível ordem?
  - Sistemática de exploração
- Duas abordagens
  - Explorar o vértice *descoberto* “mais antigo”
  - Explorar o vértice *descoberto* “mais recente”

# Busca em Largura (BFS)

- Explorar vértices descobertos mais antigos primeiro

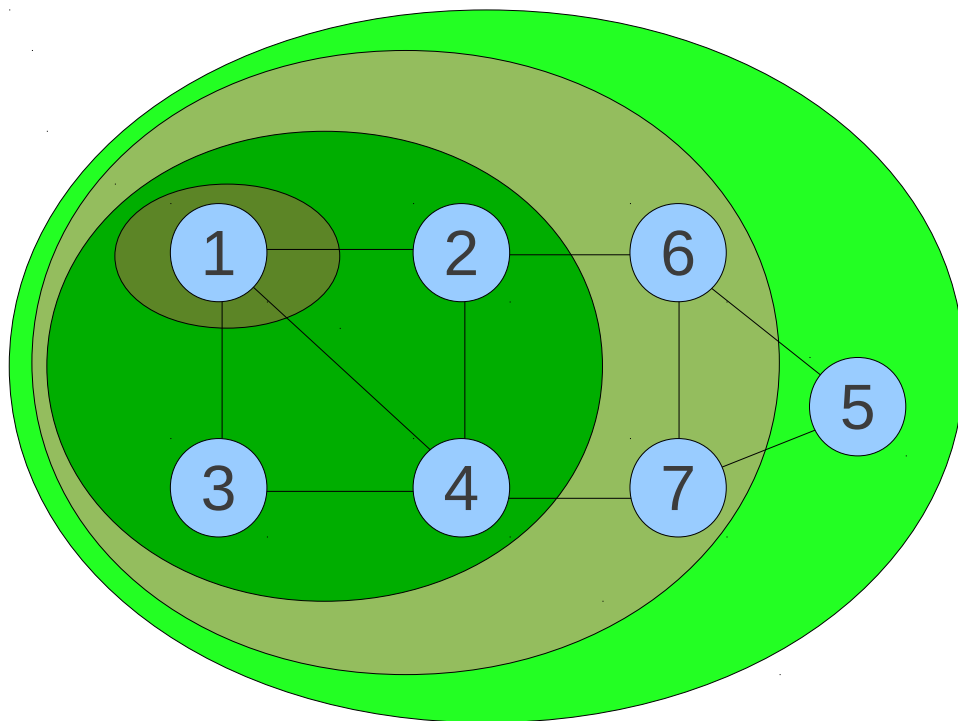


- Origem: vértice 1
- Em que ordem os vértices são *descobertos*?

**Assumir arestas são exploradas em ordem crescente dos vértices adjacentes (matriz ou lista de adjacência)**

# Interpretação

- Onda é propagada à partir da raiz
- Onda expande em círculos, descobrindo vértices alcançáveis!



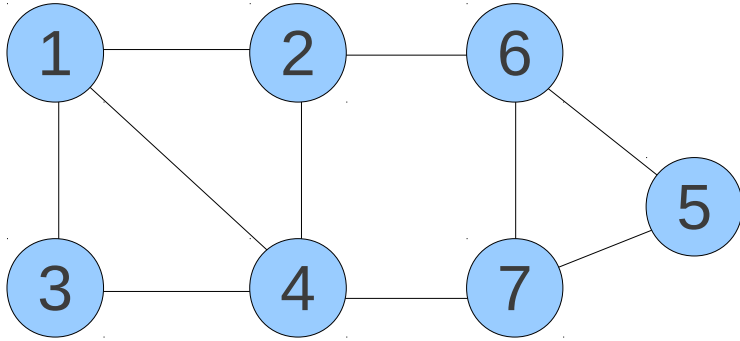
**Busca em Largura!**



# Camadas

- $L_i$  : conjunto de vértices pertencentes a camada  $i=0, 1, 2, \dots$
- $L_0$  : vértice origem
- $L_{i+1}$  : conjunto de vértices que **não fazem parte** de uma camada anterior e que **possuem uma aresta** com algum vértice da camada  $L_i$

# Camadas: Exemplo

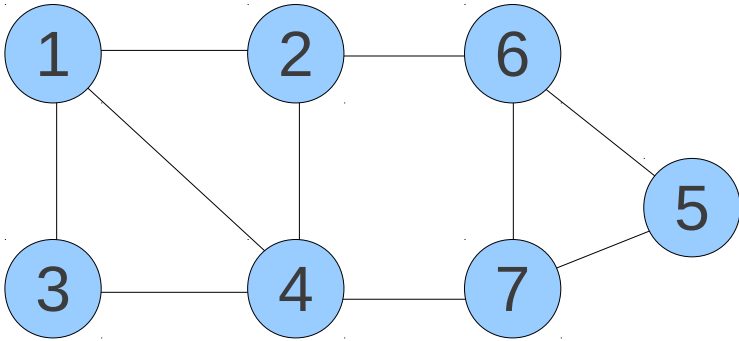


■  $L_0$  : vértice 2

■  $L_i$  : ?

# Distância

- Comprimento do **menor** caminho simples entre dois vértices
- Função  $d(u,v)$ , onde  $u$  e  $v$  são vértices
  - infinito quando não há caminho



## ■ Exemplo

■  $d(1,2) = ?$

■  $d(6, 3) = ?$

■  $d(7, 1) = ?$

# Camadas e Distância

- Qual é a relação entre eles?
- Vértices pertencentes a camada  $L_i$  têm distância  $i$  da origem!

**Busca em largura (BFS)  
calcula distância!**

# Grafo Acíclico

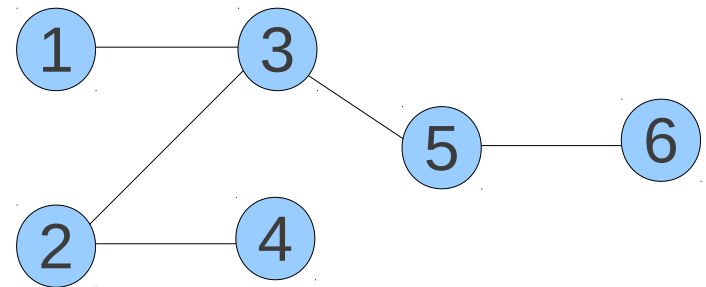
- Grafo acíclico é um grafo que não possui ciclos

- lembrem do “ciclo”?

- Exemplo:

- $K_4$  é acíclico?

É acíclico?

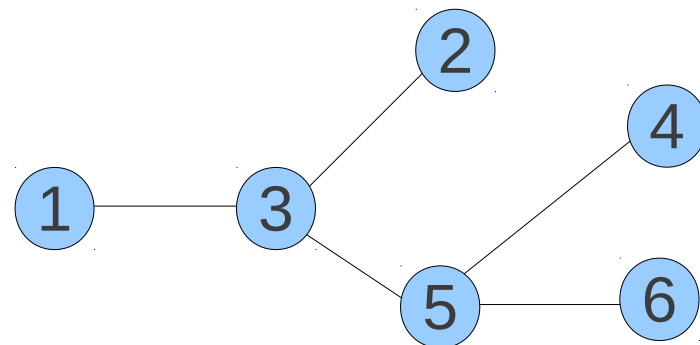


- Como descobrir se um grafo é acíclico?

**Algoritmo eficiente!**

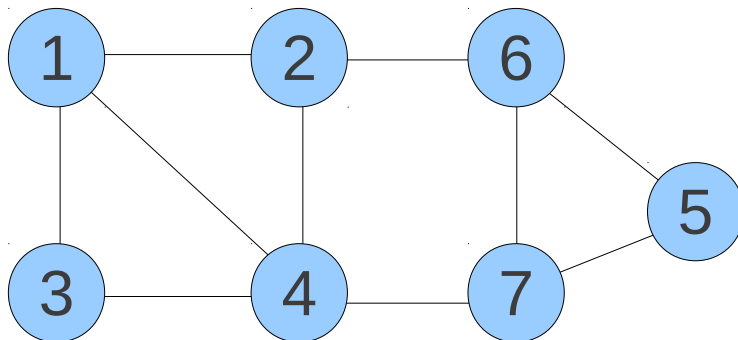
# Árvores

- Uma árvore é um grafo acíclico conexo
  - definição de árvore!
- Folha: vértice com grau 1
- Raiz: um determinado vértice
  - define orientação na árvore (pai, filhos, descendentes e acenstrais)
- Quantos caminhos (simples) distintos existe entre dois vértices quaisquer?
- Quantas arestas possui uma árvore com  $n$  vértices?

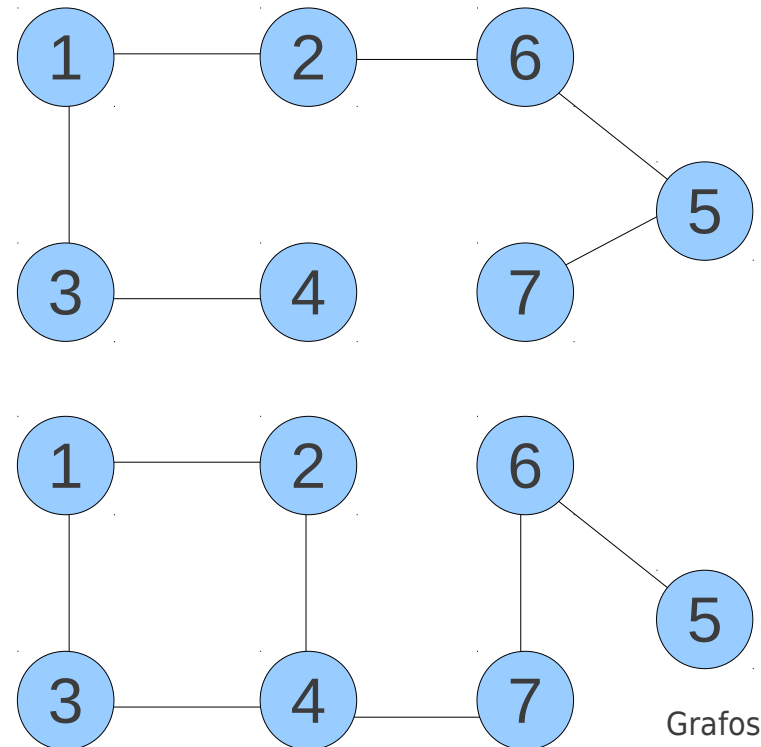


# Árvore Geradora

- Subgrafo que contém **todos** os vértices de  $G$  e é uma **árvore**
  - em inglês, “spanning tree”
  - árvore que “alcança” todos os vértices



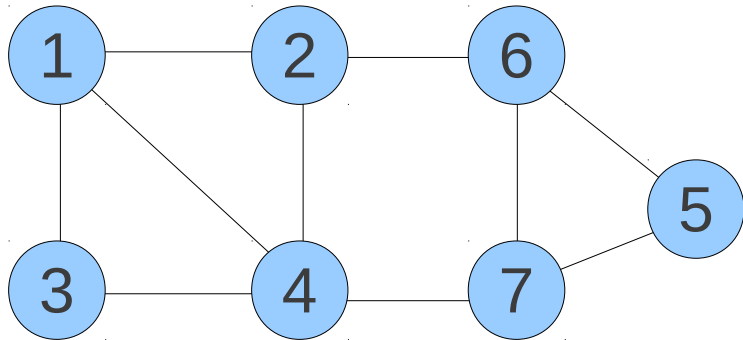
É árvore geradora?



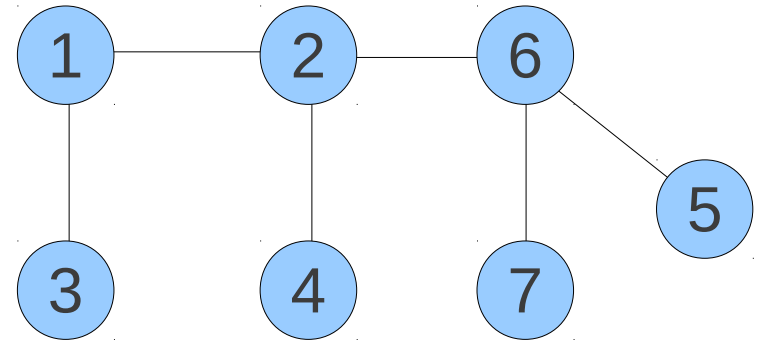
# Árvore Geradora da BFS

- Árvore *induzida* pela busca em largura
  - Raiz: vértice de origem
  - Pai de  $v$ : nó que levou à descoberta de  $v$

raiz: nó 6



Árvore geradora



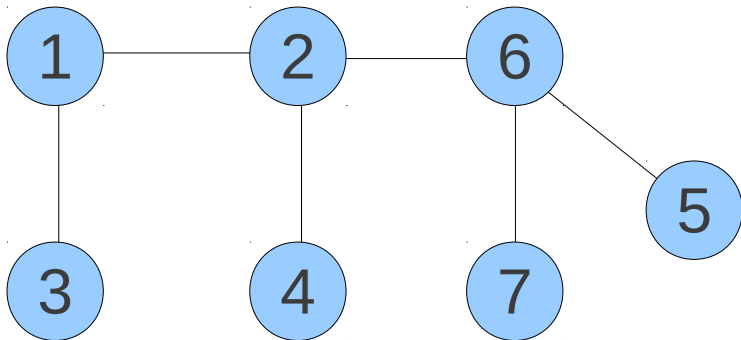
- Ordem da busca *define* árvore
- $L_i$  = nível  $i$  da árvore



# Menor Caminho

- Árvore geradora define menor caminho
- Dado vértice  $v$  (raiz) e outro vértice  $w$  qq.
  - menor caminho definido pela sequência de pais de  $w$  até a raiz

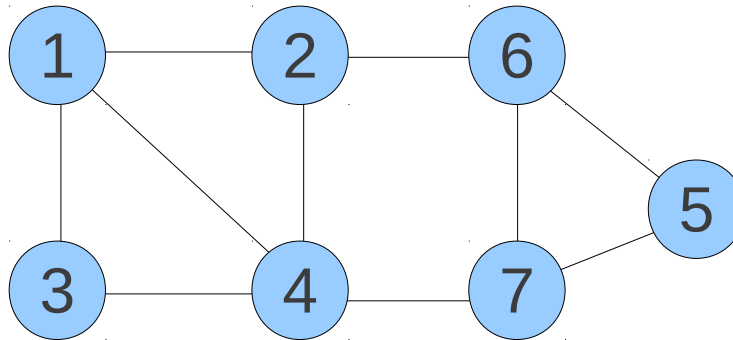
Árvore geradora (raiz 6)



- menor caminho entre 3 e 6?
- menor caminho entre 3 e 7?
- **Cuidado!** Árvore define menor caminho para raiz!

# Busca em Largura (BFS)

- Explorar vértices descobertos mais antigos primeiro



- Origem: vértice 1
- Em que ordem os vértices são *descobertos*?

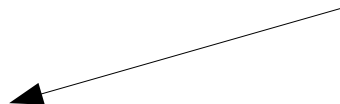
**Assumir arestas são exploradas em ordem crescente dos vértices adjacentes (matriz ou lista de adjacência)**

# Implementação

- Como implementar a busca em largura?
- Algoritmo simples, sem camadas, somente para percorrer o grafo, utilizando **fila**

1. Desmarcar todos os vértices
2. Definir fila  $Q$  vazia
3. Marcar  $s$  e inserir  $s$  na fila  $Q$
4. Enquanto  $Q$  não estiver vazia
5.     Retirar  $v$  de  $Q$
6.     Para todo vizinho  $w$  de  $v$  faça
7.         Se  $w$  não estiver marcado
8.             marcar  $w$
9.             inserir  $w$  em  $Q$

$s$  é o vértice raiz!



# Complexidade

■ Qual é a complexidade deste algoritmo?

■ utilizando lista de adjacência?

1. Desmarcar todos os vértices ← percorre os  $n$  vértices

2. Definir fila  $Q$  vazia

3. Marcar  $s$  e inserir  $s$  na fila  $Q$

4. Enquanto  $Q$  não estiver vazia ← quantos vértices em  $Q$ ?

5.       Retirar  $v$  de  $Q$

6.       Para todo vizinho  $w$  de  $v$  faça ← Percorre vizinhos do vértice, para cada vértice

7.           Se  $w$  não estiver marcado

8.           marcar  $w$

9.           inserir  $w$  em  $Q$

$$\sum_{v \in V} d(v) = 2m$$

# Complexidade

- $O(n + m)$
- Complexidade linear
  - mesma ordem de grandeza do tempo necessário para ler o grafo!
- Se grafo for denso,  $m \sim n^2$

## Recordação

- O que é  $O(n + m)$ ?
- Por que não  $O(n)$  ou  $O(m)$  ou  $O(n^2)$ ?
- $O(n + m)$  representa o **máximo** entre  $n$  e  $m$ !      ← “+” = max

# Conectividade

- **Problema:** Como saber se existe caminho entre dois vértices?
  - ex. ir do Rio à Xapuri de carro?

**Usar BFS para resolver este problema!**

- Como?
- Marca s como raiz
- Realiza BFS
- Ao terminar a BFS se t estiver marcado, então há caminho, caso contrário, não há

# Grafo Conexo

- **Problema:** como determinar se um grafo  $G$  é conexo?

## Aplicações?

- Transporte aéreo comercial
  - voar de qualquer cidade para qualquer cidade

**Idéia:** utilizar BFS!

- Escolher vértice  $v$  qualquer de  $G$
- Executar BFS à partir de  $v$
- Verificar se todos vértices foram marcados

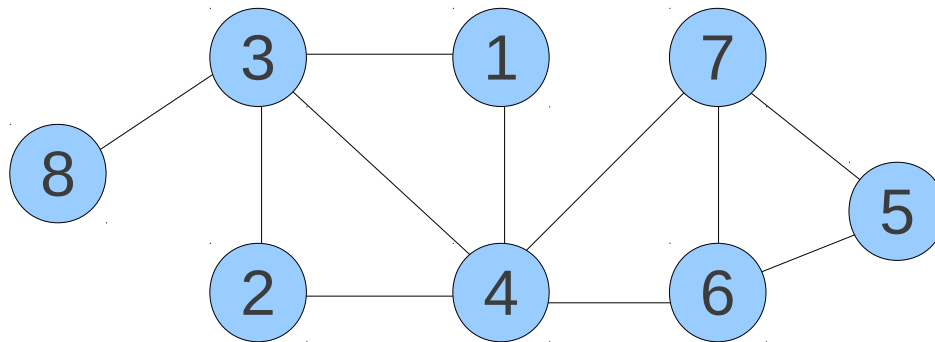
# Busca em Profundidade (DFS)

- Explorar vértices descobertos mais recentes primeiro
- Oposto de BFS: explora mais antigo primeiro
- **Interpretação**
  - procurar uma saída de um labirinto
  - vai fundo atrás da saída (tomando decisões a cada encruzilhada)
  - volta a última encruzilhada quando encontrar um beco sem saída (ou lugar já visitado)



# Busca em Profundidade

- Explorar o grafo abaixo usando DFS
  - início: vértice 4
  - vizinhos encontrados em ordem crescente



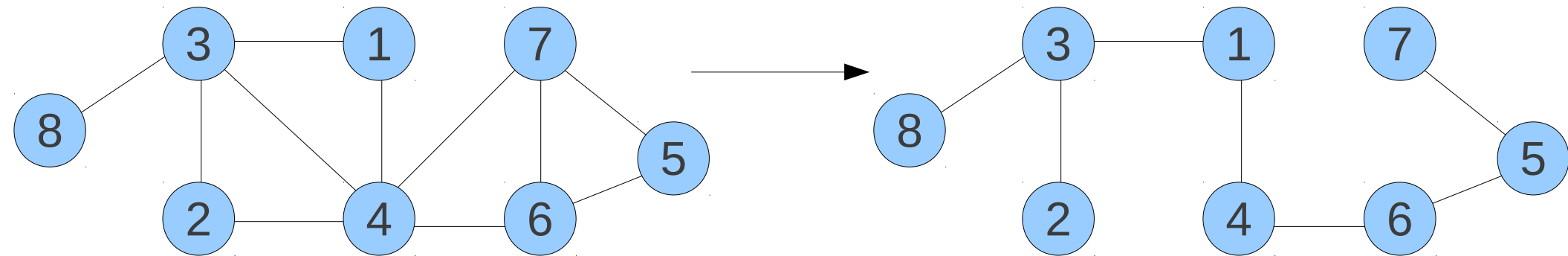
- Ordem de descobrimento dos vértices?

# Árvore de Profundidade

- Árvore *induzida* pela busca em profundidade
  - Raiz: vértice de origem
  - Pai de  $v$ : nó que levou à descoberta de  $v$

raiz: nó 4

Árvore de profundidade

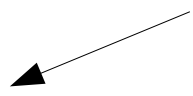


- Ordem da busca *define* árvore

# Implementação

## ■ Como implementar DFS?

### ■ utilizando algoritmo recursivo

1. DFS( $u$ )
  2. Marcar  $u$  como descoberto
  3. Para cada aresta  $(u, v)$  incidente a  $u$
  4.     Se  $v$  não estiver marcado
  5.         DFS( $v$ )   // chamada recursiva
  - 6.
  7. Desmarcar todos os vértices
  8. Escolher vértice inicial  $s$  
  9. DFS( $s$ )
- $s$  é o vértice raiz!

# Implementação

- Como implementar DFS?

- utilizando uma pilha

1. DFS( $s$ )

2. Desmarcar todos os vértices

3. Definir pilha  $P$  com um elemento  $s$

4. Enquanto  $P$  não estiver vazia

5. Remover  $u$  de  $P$  // no topo da pilha

6. Se  $u$  não estiver marcado

7. Marcar  $u$  como descoberto

8. Para cada aresta  $(u,v)$  incidente a  $u$

9. Adicionar  $v$  em  $P$  // no topo

# Complexidade

## ■ Qual é a complexidade deste algoritmo?

1. Desmarcar todos os vértices
2. Definir pilha P com um elemento s
3. Enquanto P não estiver vazia
4.     Remover u de P // no topo da pilha
5.     Se u não estiver marcado
6.         Marcar u como descoberto
7.     Para cada aresta (u,v) incidente a u
8.         Adicionar v em P // no topo

## ■ Custo para desmarcar vértices?

## ■ Quantos vértices adicionados em P?

### ■ vértice adicionado em P mais de uma vez?

## ■ Quantas vezes uma aresta é examinada?

# Complexidade

- $O(n + m)$
- Complexidade linear
  - mesma ordem de grandeza do tempo necessário para ler o grafo!
- Se grafo for denso,  $m \sim n^2$
- Mesma complexidade que BFS!

# Árvore de Profundidade

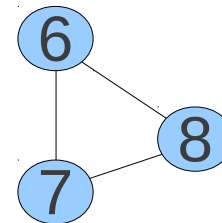
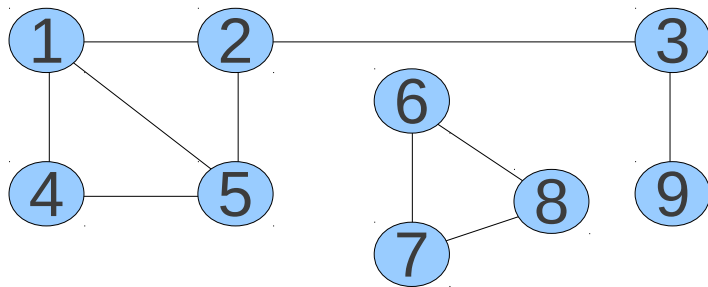
- Como obter árvore geradora induzida pela busca?
- Como modificar algoritmo para gerar a árvore?

## ■ Idéia

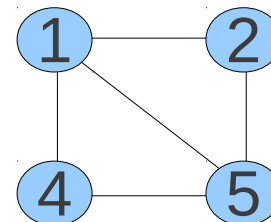
- Utilizar vetor `pai[]` para indicar o pai de um vértice  $v$  na árvore
- Idéia independe do tipo de busca (BFS, DFS, etc)

# Componentes Conexos

- Maiores subgrafos “conectados” de um grafo
  - mais precisamente...
- Subgrafos maximais de  $G$  que sejam conexos
  - *maximal*: subconjunto que maximiza a propriedade, no caso subgrafo conexo
- Exemplo:



Componente  
conexo?

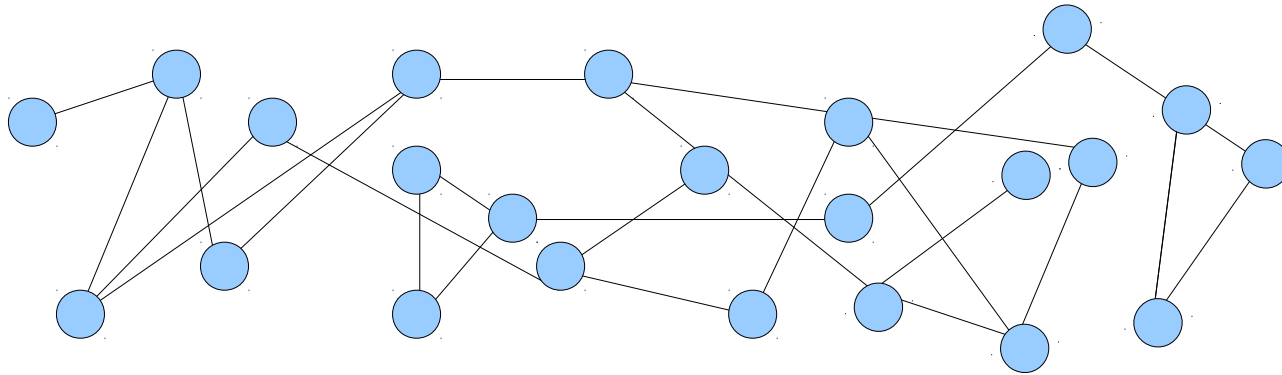


Componente  
conexo?



# Componentes Conexos

- **Problema:** Determinar o número de componentes conexos de um grafo
  - e tamanho de cada componente



**Algoritmo eficiente para resolver este problema?**

# Componentes Conexos

## Usar Busca em Grafos (BFS ou DFS)

- Desmarcar todos os vértices
- Escolher vértice s qualquer
- Realizar BFS
- Vértices marcados determinam uma CC
- Escolher vértice s não marcado qualquer
- Realizar BFS
- Vértices marcados determinam outra CC
- ...

# Componentes Conexos

## Complexidade?

- Complexidade do algoritmo anterior
- Maior número de CC de um grafo?
- Custo para detectar cada CC?
- Usar marcações diferentes para cada CC

**$O(m + n)$**