

# Teoria dos Grafos

## Aula 6

### Aula passada

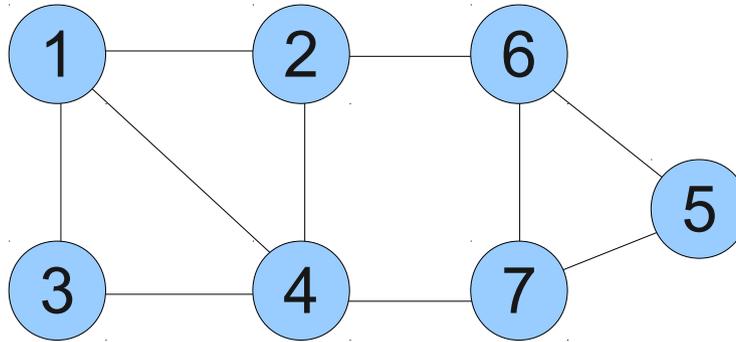
- Busca em grafos
- Busca em largura (BFS - Breadth-First Search)
- Propriedades

### Aula de hoje

- BFS – implementação
- Complexidade
- Busca em profundidade (DFS)
- Conectividade, componentes

# 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  $Q$

$s$  é o vértice raiz!

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  $Q$

# 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!

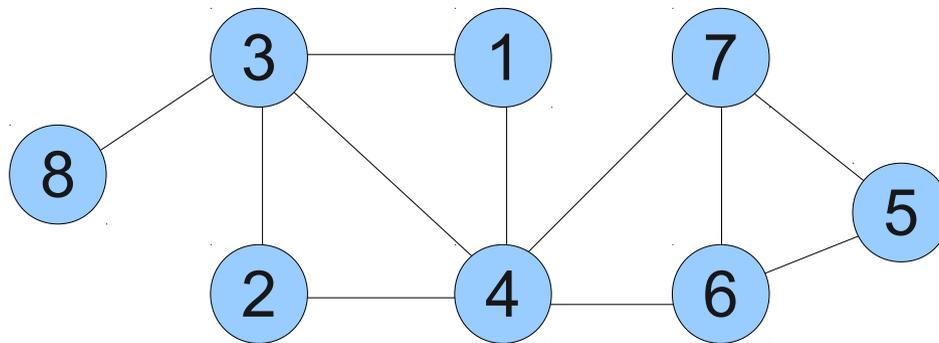
- Escolher vértice  $v$  qualquer de  $G$
- Executar BFS à partir de  $v$
- Verificar se todos vértices foram marcados

# 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



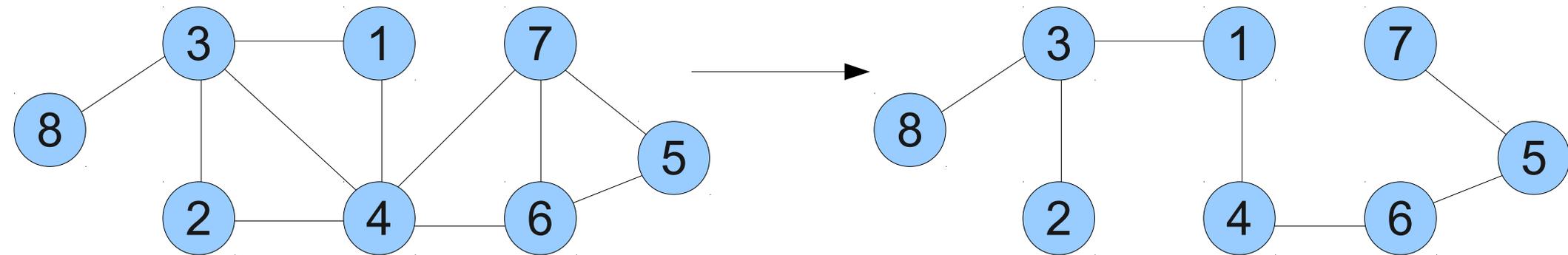
- Ordem de descobrimento dos vértices?

# 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 s é o vértice raiz!

8. Escolher vértice inicial  $s$  

9. DFS( $s$ )

# 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!

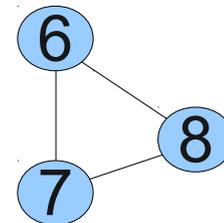
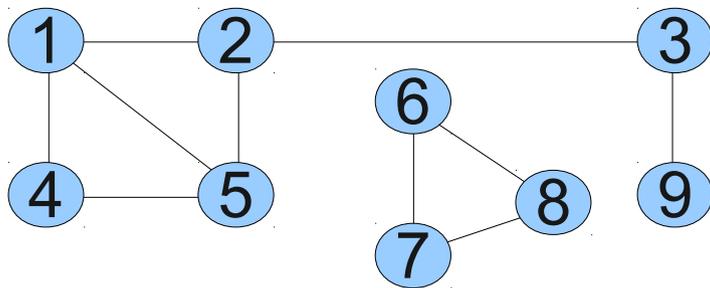
# Árvore de Profundidade

- 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 independente do tipo de busca (BFS, DFS, etc)

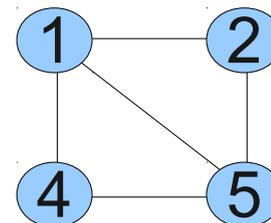
# Componentes Conexos

- 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:



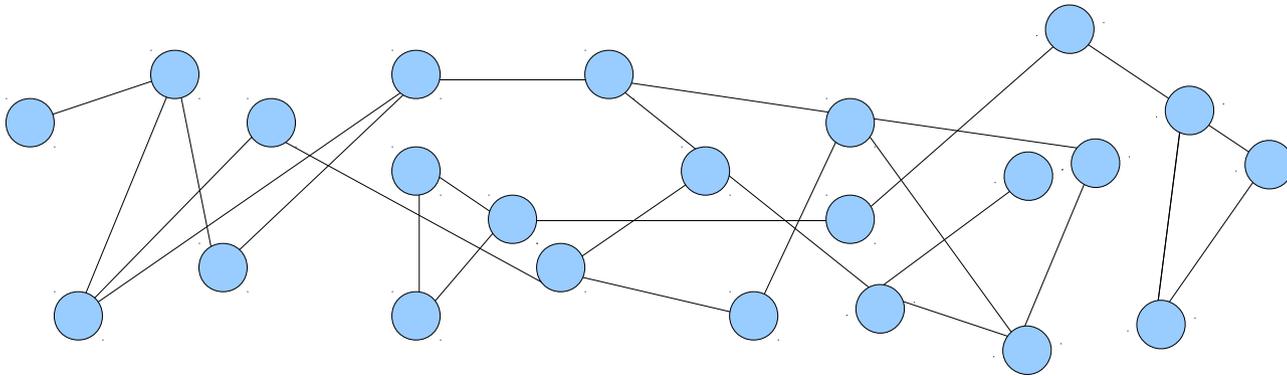
Componente  
conexo?



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

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