PANC: Projeto e Análise de Algoritmos

Aula 13: Grafos

Breno Lisi Romano

http://sites.google.com/site/blromano

Instituto Federal de São Paulo – IFSP São João da Boa Vista
Bacharelado em Ciência da Computação – 3º Semestre

INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SÃO PAULO
Campus São João da Boa Vista

Sumário

- Introdução
- Exemplos
- Terminologia
- Noções Básicas de Grafos
- Representações Computacionais de Grafos
- Busca em Grafos
  - Busca em Profundidade
  - Busca em Largura

Grafos - Motivações

- Grafos são estruturas abstratas que podem modelar diversos problemas do mundo real
- Por exemplo, um grafo pode representar conexões entre cidades por estradas ou uma rede de computadores
- O interesse em estudar algoritmos para problemas em grafos é que conhecer um algoritmo para um determinado problema em grafos pode significar conhecer algoritmos para diversos problemas reais

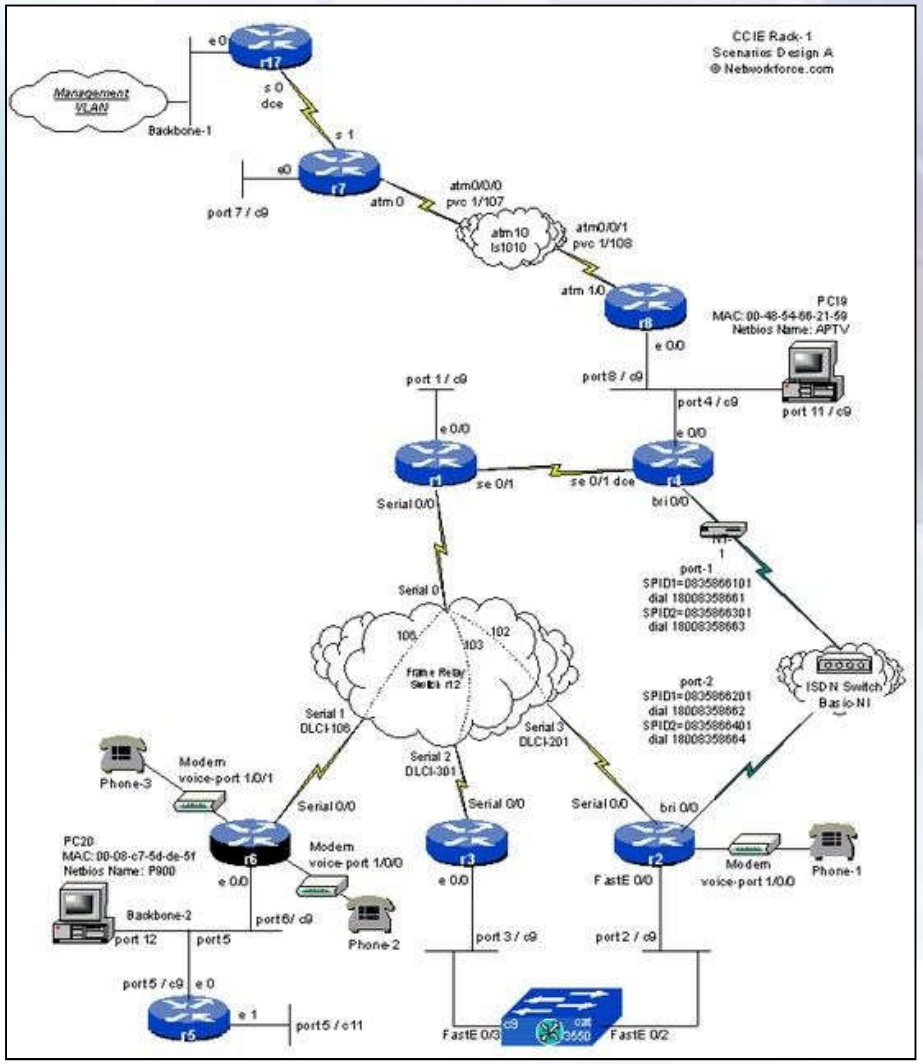
Exemplos de Grafos – Mapa Rodoviário do Estado de SP

Exemplos de Grafos – Estações de Metrô da cidade de São Paulo

Exemplos de Grafos – Rede de Relacionamentos

Exemplos de Grafos – Mapa Hidrográfico do Rio Amazonas

Exemplos de Grafos – Projeto de Redes de Computadores



Algoritmos em Grafos - Aplicações

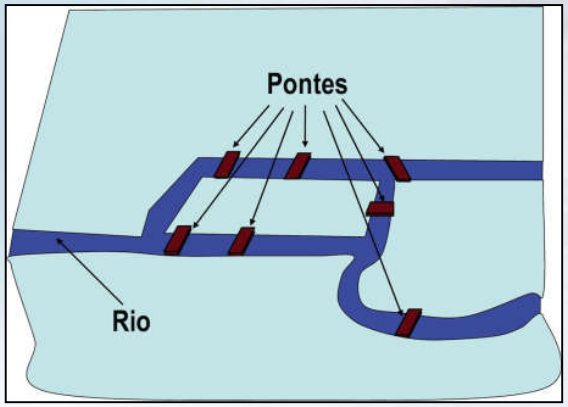
- **Caminho mínimo:** dado um conjunto de cidades, as distâncias entre elas e duas cidades A e B, determinar um caminho (trajeto) mais curto de A até B
- **Árvore Geradora de Peso Mínimo:** dado um conjunto de computadores, onde cada par de computadores pode ser ligado usando uma quantidade de fibra ótica, encontrar uma rede interconectando-os que use a menor quantidade de fibra ótica possível
- **Emparelhamento máximo:** dado um conjunto de pessoas e um conjunto de vagas para diferentes empregos, onde cada pessoa é qualificada para certos empregos e cada vaga pode ser ocupada por uma pessoa, encontrar um modo de empregar o maior número possível de pessoas
- **Problema do Caixeiro Viajante:** dado um conjunto de cidades, encontrar um passeio que sai de uma cidade, passa por todas as cidades e volta para a cidade inicial tal que a distância total a ser percorrida seja menor possível
- **Problema Chinês do Correio:** dado o conjunto das ruas de um bairro, encontrar um passeio que passa por todas as ruas voltando ao ponto inicial tal que a distância total a ser percorrida seja menor possível

Grafos – História (1)

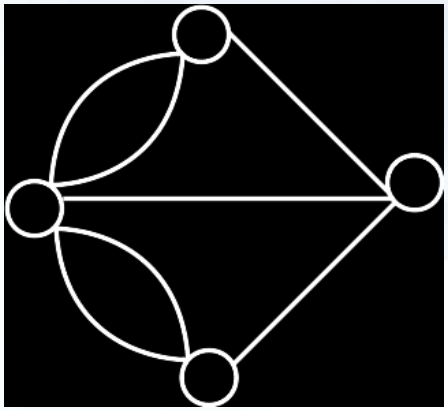
- Um grafo é uma estrutura de abstração muito útil na representação e solução de problemas computacionais, por representarem relações de interdependência entre elementos de um conjunto
- O primeiro registro de uso data de 1736, por Euler
- O problema era encontrar um caminho circular por Königsberg (atual Kaliningrado) usando cada uma das pontes sobre o rio Pregel (o Pregolya, Pregola) exatamente uma vez

Grafos – História (2)

- **1736: Euler e as Pontes de Konigsberg:**
  - **Pergunta:** Partindo de uma das margens, pode-se encontrar um percurso que passe somente **uma vez em cada ponte** e retorne ao ponto de partida?

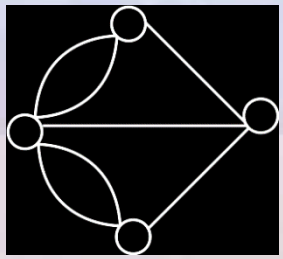


  - **Modelo Proposto por Euler para tentar resolver o problema:**
    - Cada vértice é uma área de terra firme
    - Cada aresta representa uma ponte



Grafos – História (2)

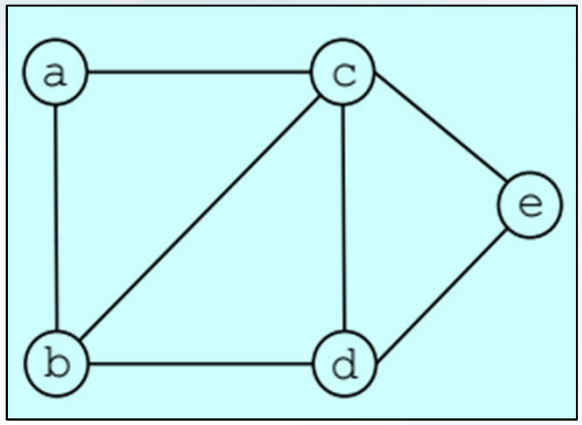
- **1736: Euler e as Pontes de Konigsberg:**
  - Se houvesse solução, ela deveria partir de um vértice qualquer e passar pelas arestas que o conectam
    - Cada vez que ele passa por uma aresta conectada a este vértice, temos:
      1. Se estávamos no vértice, então saímos dele
      2. Se não estávamos no vértice, então entramos nele



  - Como saímos de um vértice, temos que se o número de vezes que passamos por arestas conectadas ao vértice for ímpar, então terminaremos fora deste vértice
    - Por outro lado, se o número de vezes que passamos por arestas conectadas ao vértice for par, voltamos a este vértice
  - Observe então que no problema proposto, todos os vértices possuem um número ímpar de arestas, e o problema pede que comece e termine no mesmo vértice
    - Se passamos por todas as arestas, então passamos por uma quantidade ímpar de arestas conectadas àquele vértice, logo devemos estar fora do vértice → Não existe solução para este problema
  - Caminhos que viabilizam sair de um vértice, percorrer todas as arestas apenas uma vez cada e retornar para o vértice original, são chamados de Grafos Eulerianos

Grafos: Definição

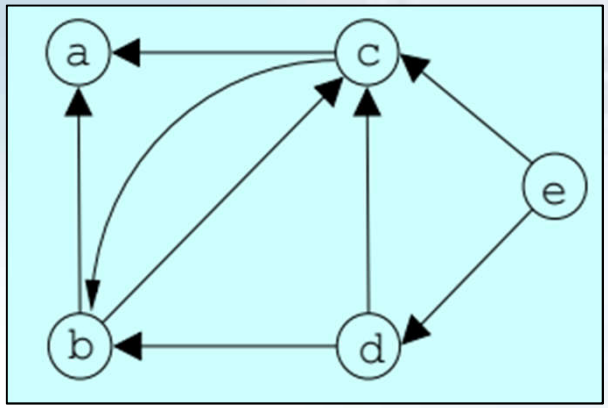
- **Definição Formal:**
  - **Grafo é um par G = (V, A), em que:**
    - **V** é um conjunto finito com **n** vértices ou nós → **V = {a, b, c, d, e}**
    - **A** conjunto finito de pares não-ordenados de vértices chamados de arestas ou arcos → **A = {(a, b), (a, c), (b, c), (b, d), (c, d), (c, e), (d, e)}**



Grafos Orientados

- **Um grafo orientado (direcionado)** é definido de forma semelhante**, com a** diferença que as **arestas consistem de pares ordenados de vértices**

- **Exemplo:**

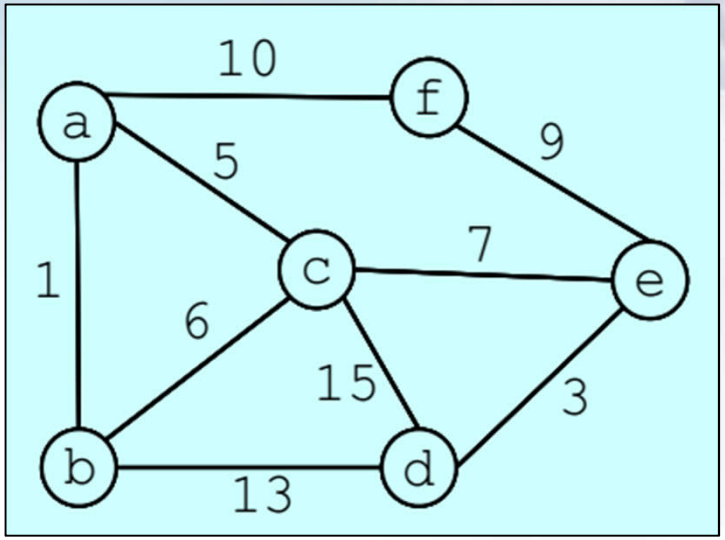


- As vezes, para enfatizar, dizemos grafo não-orientado em vez de simplesmente grafo

Grafos Ponderados

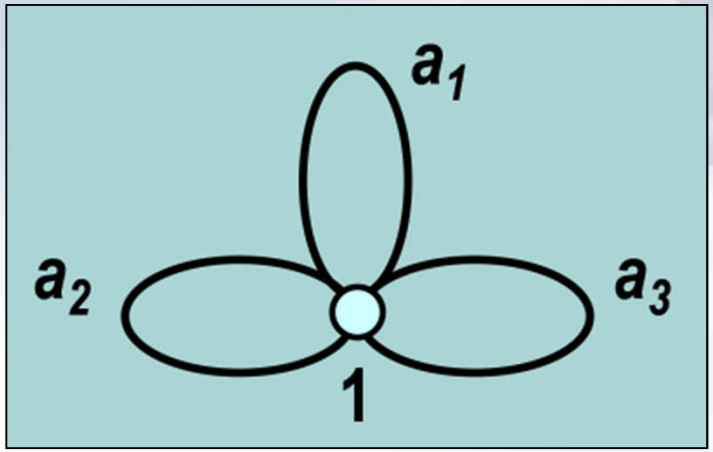
- Um **grafo** (orientado ou não) **é ponderado** se cada aresta **a** o grafo está associado um valor real **c(a)**, o qual denominamos **custo (ou peso)** da aresta

- **Exemplo:**



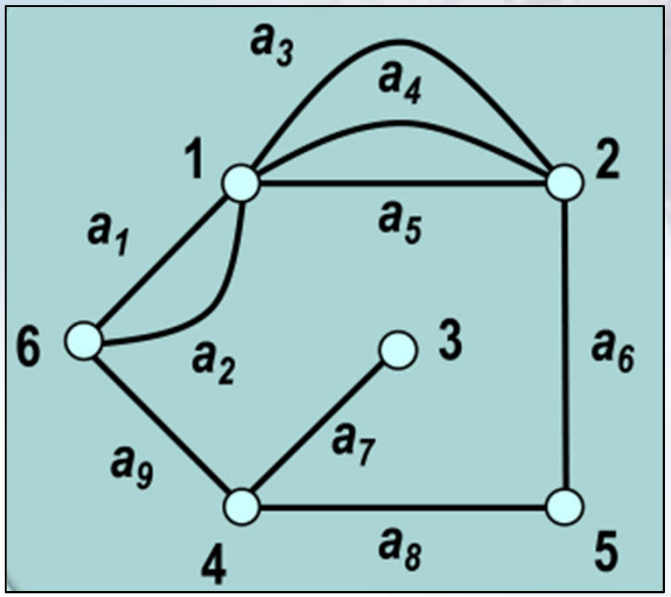
Grafos: Terminologia (1)

- **Laços:**
  - Uma aresta cujas duas extremidades incidem em um mesmo vértice



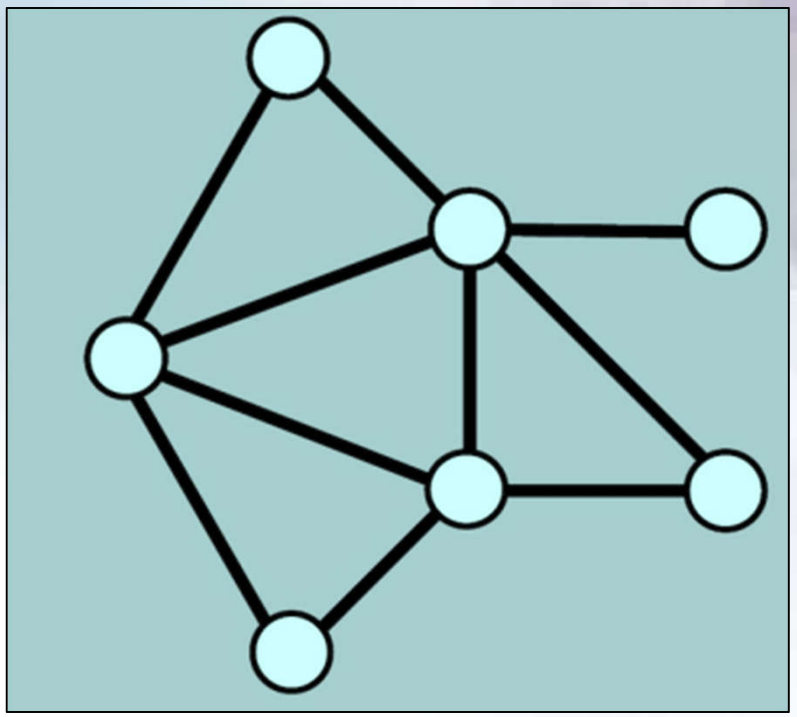
Grafos: Terminologia (2)

- **Arestas Paralelas:**
  - Mais de uma aresta associada ao mesmo par de vértices



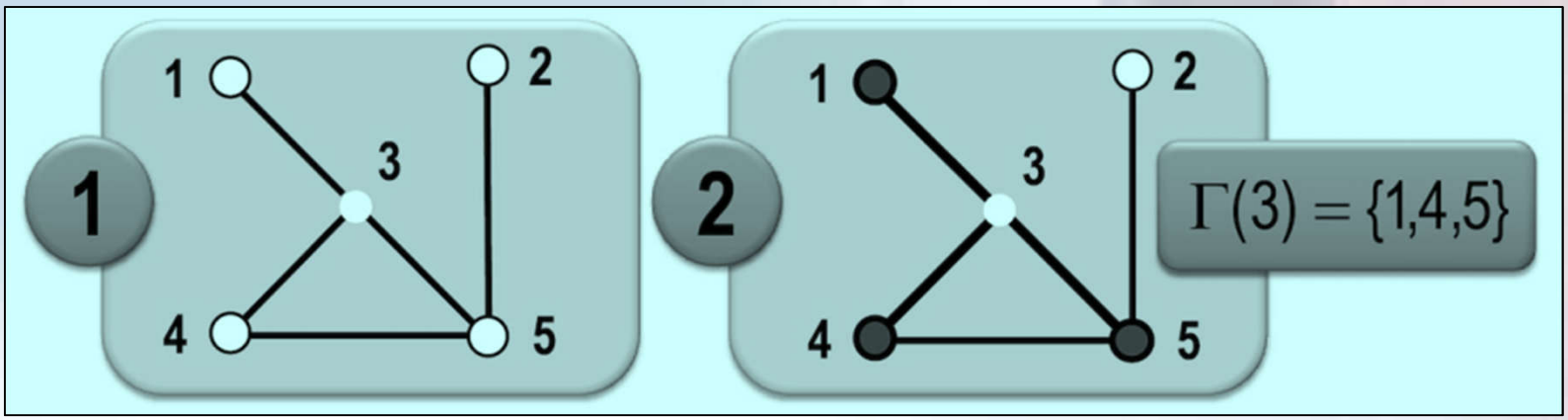
Grafos: Terminologia (3)

- **Grafo Simples:**
  - Grafo que não possui laços e nem arestas paralelas



Grafos: Terminologia (4)

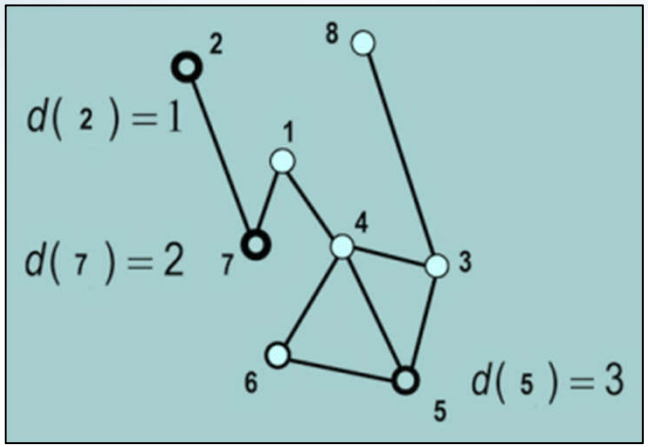
- **Vértices Adjacentes:**
  - Vértices que são os pontos finais de uma mesma aresta
  - A função Γ(i) retorna o conjunto de vértices adjacentes ao vértice i



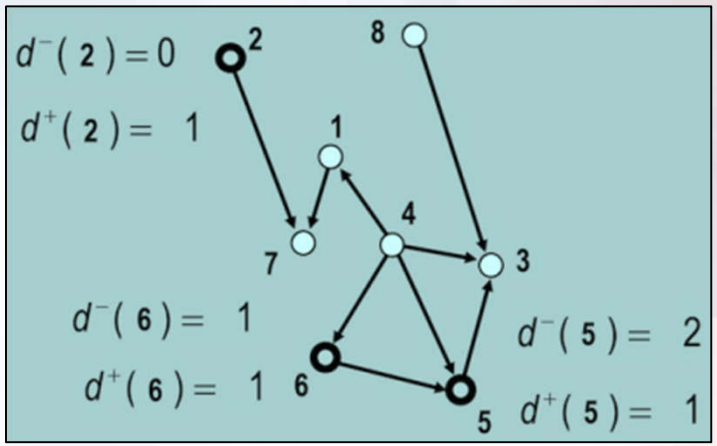
Grafos: Terminologia (5)

- **Grau de um Vértice:**
  - O grau (d(i)) de um vértice i em um grafo não direcionado é igual o número de arestas incidentes a i
  - O grau de entrada ($d^-(i)$) de um vértice i em um grafo direcionado é igual o número de arestas que entram em i
  - O grau de saída ($d^+(i)$) de um vértice i em um grafo direcionado é igual o número de arestas que saem de i

Grafo Não Direcionado

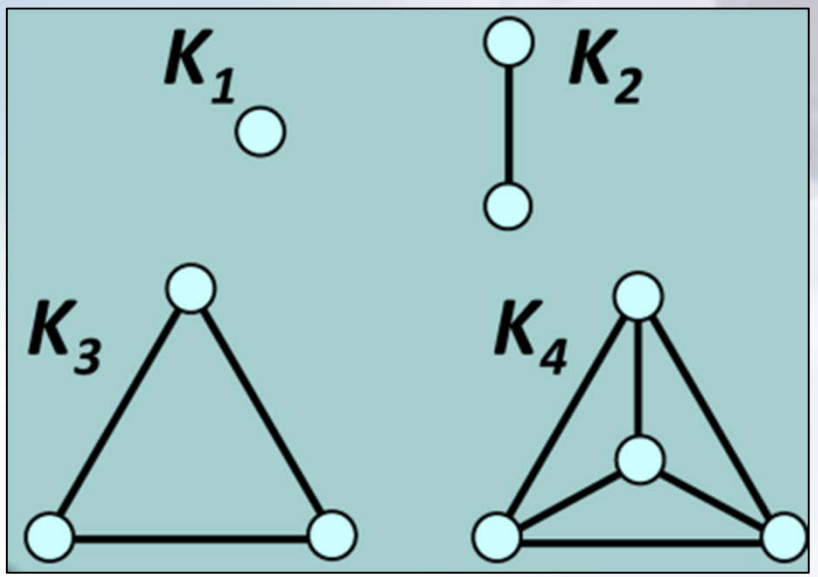


Grafo Direcionado



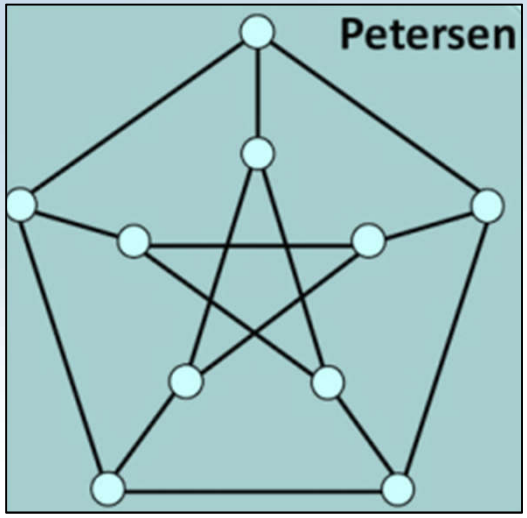
Grafos: Terminologia (6)

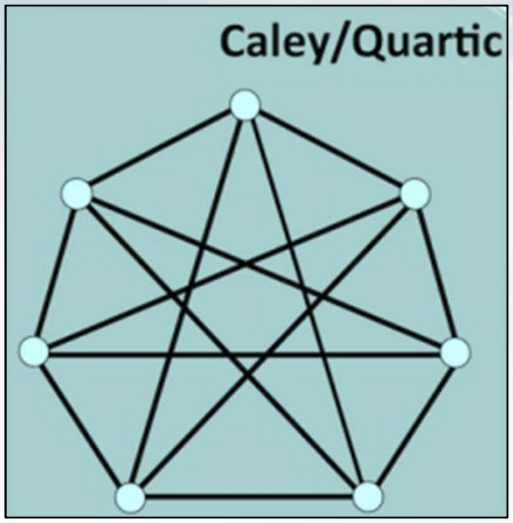
- **Grafo Completo:**
  - Um grafo completo com n vértices, denominado K_n é um grafo simples contendo exatamente uma aresta para cada par de vértices distintos

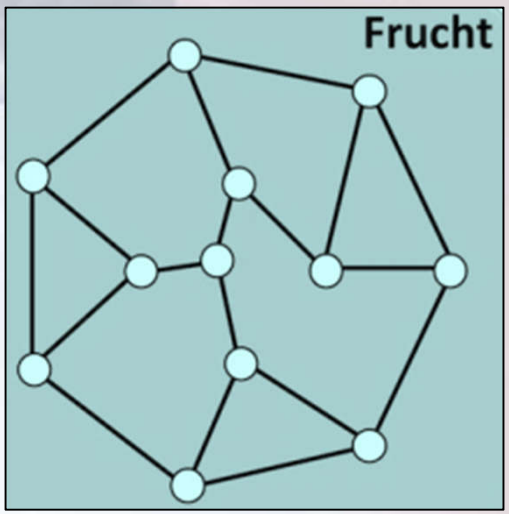


Grafos: Terminologia (7)

- **Grafo Regular:**
  - Grafo no qual todos os vértices possuem o mesmo grau
  - Obs: qualquer grafo completo é regular

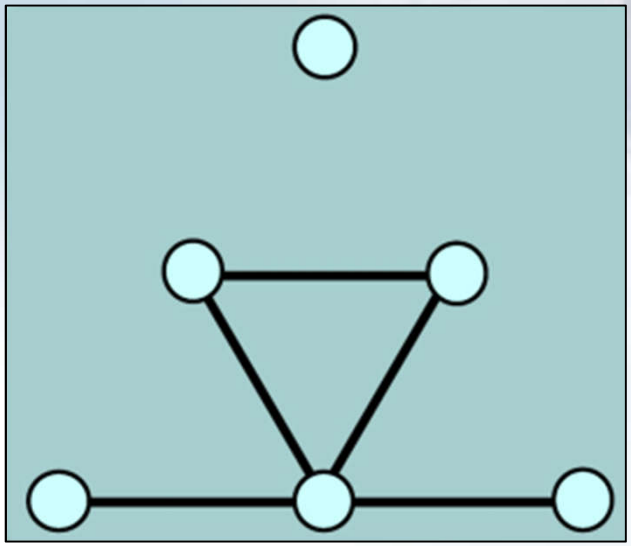






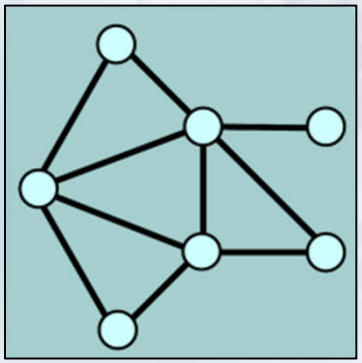
Grafos: Terminologia (8)

- **Vértice Isolado:**
  - Vértice com nenhuma aresta incidente

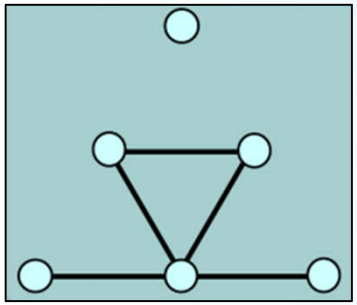


Grafos: Terminologia (9)

- **Grafo Conexo:**
  - Para todo par de vértices i e j de G existe pelo menos um caminho entre i e j

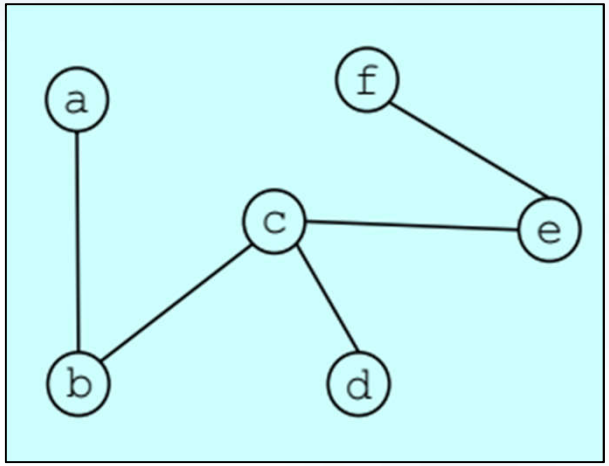


- **Grafo Desconexo:**
  - Consiste de 2 ou mais grafos conexos, chamados de componentes



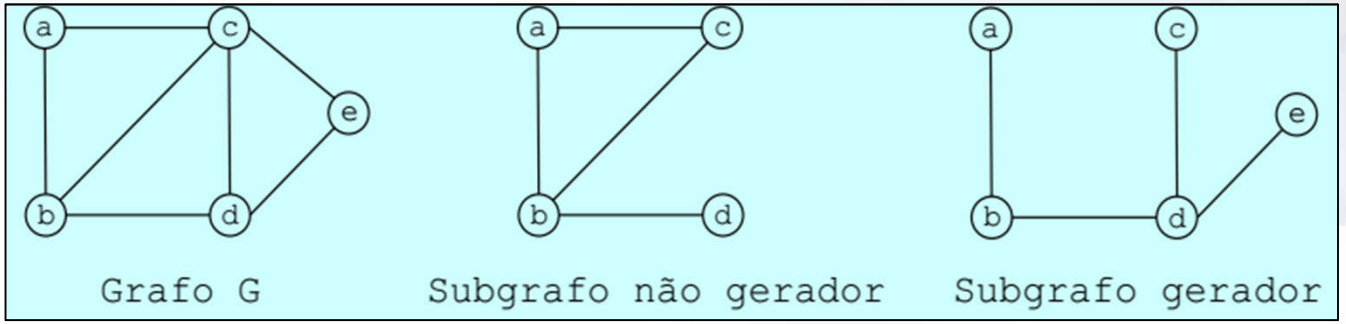
Grafos: Terminologia (10)

- **Árvore:**
  - Um grafo G é uma árvore se é conexo e acíclico.
  - As seguintes afirmações são equivalentes:
    - G é uma árvore
    - G é conexo e possui exatamente |V| - 1 arestas
    - G é conexo e a remoção de qualquer aresta desconecta o grafo
    - Para todo par de vértices (v_1 e v_2) de G, existe um único caminho de v_1 a v_2 em G



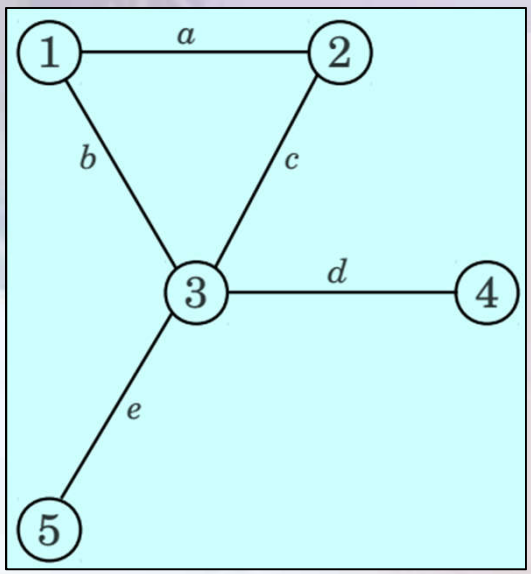
Grafos: Terminologia (11)

- **Subgrafos:**
  - **Definição:** Um grafo $G_s = (V_s, A_s)$ é dito ser um subgrafo de um grafo $G = (V, A)$ se todos os vértices e todas as arestas de G_s estão em G, ou seja, se $V_s \subseteq V$ e $A_s \subseteq A$
    - Um **subgrafo gerador** de G é um subgrafo G_s com $V_s = V$
  - **Observações:**
    - Todo grafo é subgrafo de si próprio
    - O subgrafo G_{s2} de um subgrafo G_s de G também é subgrafo de G
    - Um vértice simples de G é um subgrafo de G
    - Uma aresta simples de G (juntamente com suas extremidades) é um subgrafo de G



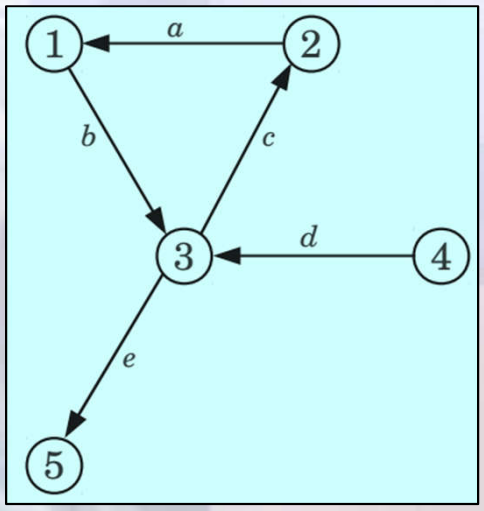
Grafos: Terminologia (12)

- **Passeio:**
  - Um passeio é uma sequência finita de vértices e arestas
  - Cada vértice da sequência é incidente a aresta que o precede e a aresta seguinte
  - Essa sequência deve acabar e iniciar em um vértice (não necessariamente os mesmos)
    - Ex.: 1 - a - 2 - c - 3 - d - 4 - d - 3 - e – 5 ou: 1 - 2 - 3 - 4 - 3 - 5
  - O passeio pode ser:
    - **Aberto:** quando inicia e acaba em vértices diferentes (o caso acima)
    - **Fechado:** quando inicia e acaba no mesmo vértice. Ex.: 1-2-3-4-3-5-3-1

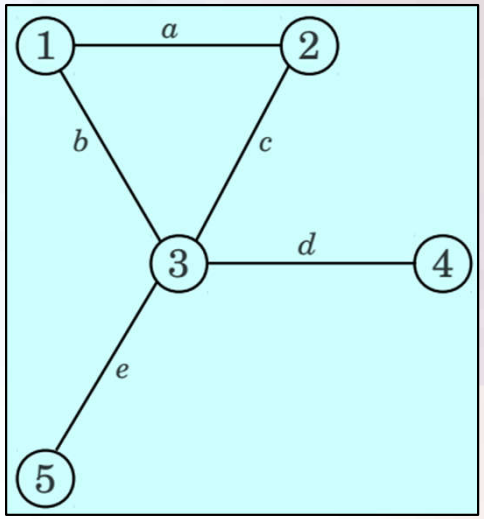


Grafos: Terminologia (13)

- **Cadeia:**
  - Um passeio que não repete arestas
  - Exemplo: Ex.: 4 - 3 - 2 - 1 - 3 - 5

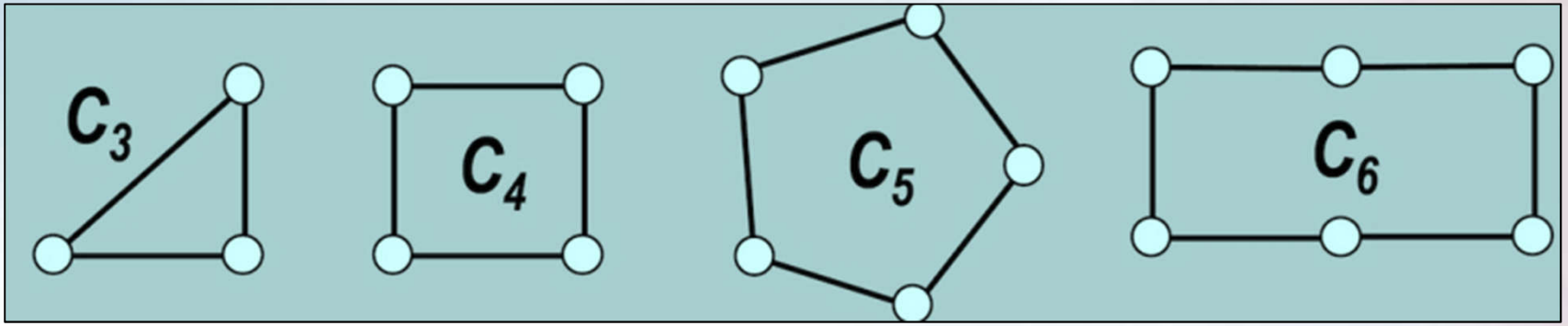


- **Caminho:**
  - Uma cadeia sem repetição de vértices
  - Pode ser:
    - **Aberto**: quando inicia e acaba em vértices diferentes. Ex.: 1-2-3-5
    - **Fechado**: quando inicia e acaba no mesmo vértice. Ex.: 1-2-3-1
  - **Comprimento**: o comprimento de um caminho é o número de arestas que o mesmo inclui



Grafos: Terminologia (14)

- **Ciclos:**
  - Um ciclo é um caminho fechado
  - Alguns autores, utilizam o termo circuito para o caso de grafos orientados
  - Grafo Ciclo: Um grafo ciclo C_n é um grafo com n vértices formado por apenas um ciclo passando por todos os vértices



Alguns Exemplos de Grafos

- **Floresta:** grafo acíclico (não precisa ser conexo). Cada componente é uma árvore
- **Grafo completo:** para todo par de vértices u e v, a aresta (u, v) pertence ao grafo
- **Grafo bipartido:** possui uma bipartição (A, B) do conjunto de vértices tal que toda aresta tem um extremo em A e outro em B
- **Grafo planar:** pode ser desenhado no plano de modo que arestas se interceptam apenas nos extremos

Grafos – Representações Computacionais

- A **complexidade dos algoritmos** para **soluções de problemas modelados por grafos** depende fortemente da sua **representação interna**

- Existem **duas representações** canônicas:
  - Matriz de Adjacência
  - Listas de Adjacência

- O uso de uma ou de outra num determinado algoritmo depende da natureza das operações que ditam a complexidade do algoritmo

Grafos – Matriz de Adjacências (1)

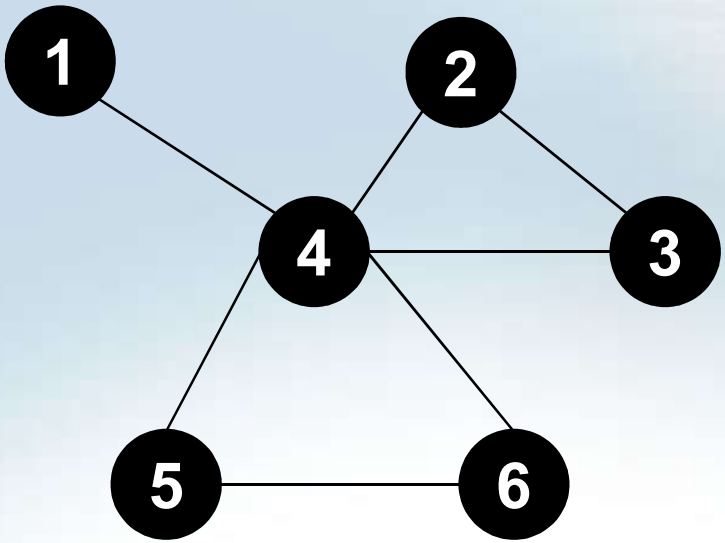
- **Matriz A_{nxn}**, sendo que:

$$a_{ij} = \begin{cases} 1 & \text{se existe a aresta/arco } (v_i, v_j) \\ 0 & \text{caso contrário} \end{cases}$$

- **Propriedades:**
  - Simétrica para grafos não direcionados
  - Consulta existência de uma aresta/arco com um acesso à memória: O(1)
  - Ocupa $\Theta(n^2)$ de espaço mesmo para grafos esparsos

Grafos – Matriz de Adjacências (2)

- Para o **grafo não ordenado,** definir a **matriz de adjacência:**

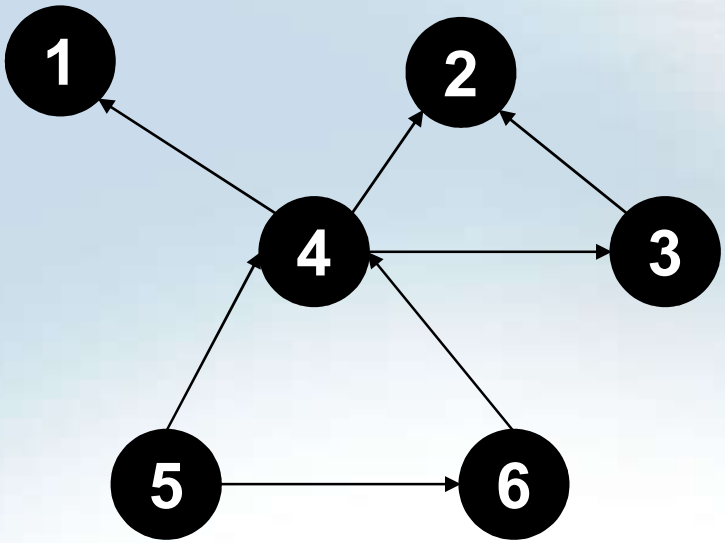


Nós	1	2	3	4	5	6
1	0	0	0	1	0	0
2	0	0	1	1	0	0
3	0	1	0	1	0	0
4	1	1	1	0	1	1
5	0	0	0	1	0	1
6	0	0	0	1	1	0

Matriz de Adjacência

Grafos – Matriz de Adjacências (3)

- Para o **grafo ordenado,** definir a **matriz de adjacência:**



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

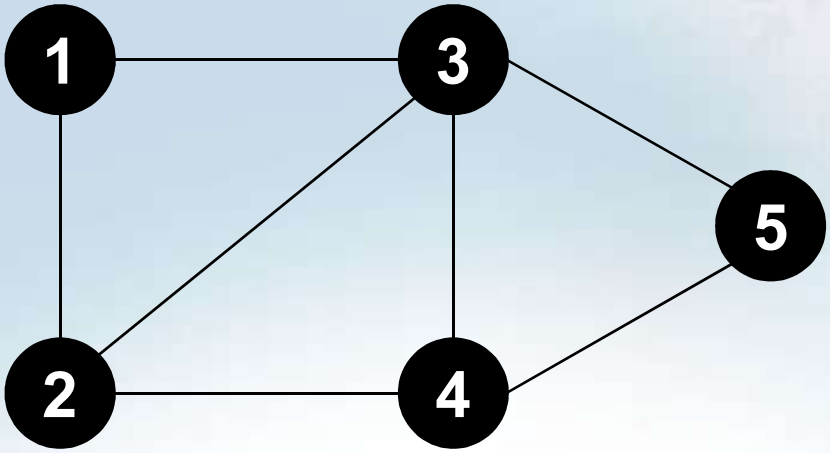
Matriz de Adjacência

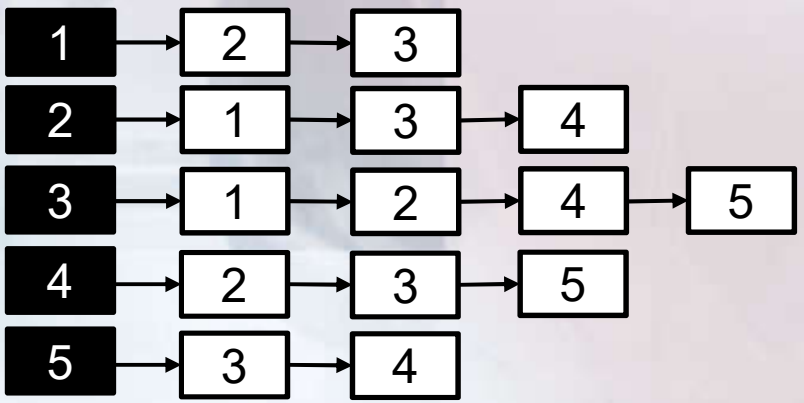
Grafos – Lista de Adjacências (1)

- Usa n listas, uma para cada vértice
- Lista de v_i (o *i*-ésimo vértice) contém todos os vértices adjacentes a ele
  - Para cada vértice v, temos uma lista ligada Adj[v] dos vértices adjacentes a v, ou seja, w aparece em Adj[v] se (v, w) é uma aresta de G
    - Os vértices podem estar em qualquer ordem na lista
- **Propriedades:**
  - Ocupa menos memória: O(m)
  - No entanto, a complexidade da operação para determinar uma adjacência é limitada por O(n)

Grafos – Lista de Adjacências (2)

- Para o **grafo não ordenado,** definir a **lista de adjacência:**

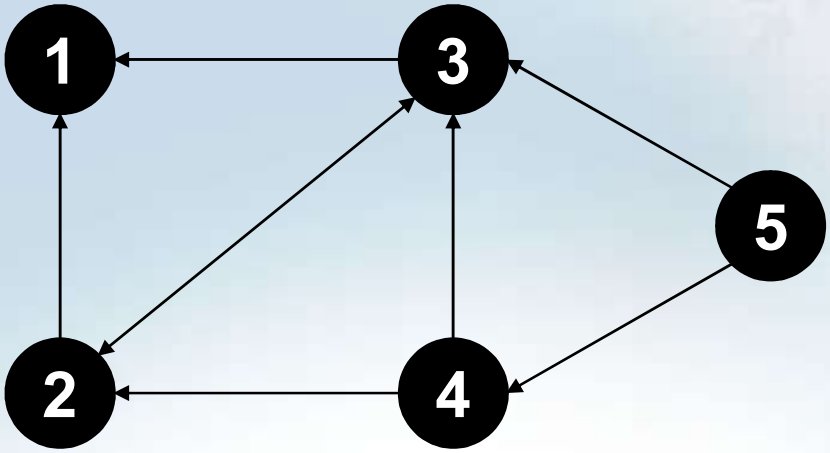


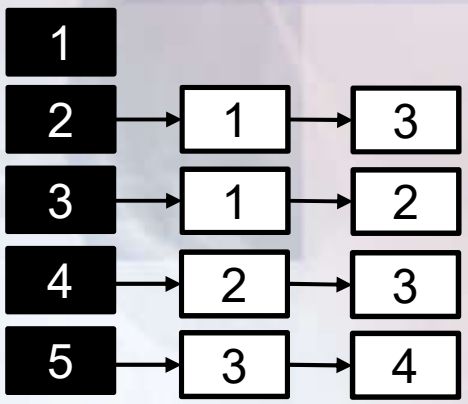


Lista de Adjacência

Grafos – Lista de Adjacências (2)

- Para o **grafo ordenado,** definir a **lista de adjacência:**





Lista de Adjacência

Grafos – Matriz x Lista de Adjacências

- Quanto a **pesquisa**:
  - **Matriz de Adjacência:** é fácil verificar se (i, j) é uma aresta de G
  - **Lista de Adjacência:** é fácil descobrir os vértices adjacentes a um dado vértice (ou sejam listar Adj[v])

- Quanto ao **espaço**:
  - **Matriz de Adjacência: espaço Θ($|V|^2$)** → Adequada grafos densos
  - **Lista de Adjacência: espaço Θ(|V|+|A|)** → Adequada a grafos esparsos

Grafos – Extensões das Representações

- Existem outras alternativas para representar grafos, mas matrizes e listas de adjacências são as mais utilizadas

- Elas podem ser adaptadas para representar grafos ponderados, grafos com laços e arestas múltiplas, grafos com pesos nos vértices etc.

- Para determinados problemas, é essencial ter estruturas de dados adicionais para melhorar a eficiência dos algoritmos

Estudando...

BUSCA EM GRAFOS

Busca em Grafos

- Dependendo do critério utilizado para escolha dos vértices e arestas a serem visitados, diferentes tipos de buscas são desenvolvidos a partir da busca genérica

- Grafos são estruturas mais complicadas do que listas, arrays e árvores binárias → Precisa-se de métodos para explorar/percorrer um grafo

- Basicamente, **duas buscas completas** em grafos são essenciais:
  - **Busca em Profundidade** (ou **DFS** – *Depth-First Search*)
  - **Busca em Largura** (ou **BFS** – *Breadth-First Search*)

- Para denotar **complexidades** nas expressões com O ou Θ, utiliza-se V e A em vez de |V [G]| ou |A[G]|
  - Por exemplo: $\Theta(V + A)$ ou $O(V^2)$

Busca em Profundidade – DFS

- A Busca em Profundidade **visita todos os vértices de um grafo**, usando como **critério os vizinhos do vértice visitado mais recentemente**
  - A estratégia consiste em pesquisar o grafo o mais "profundamente" sempre que possível
- Aplicável tanto a grafos orientados quanto não orientados
- Possui um número enorme de aplicações:
  - Determinar os componentes de um grafo
  - Ordenação topológica
  - Determinar componentes fortemente conexos
  - Subrotina para outros algoritmos
- **Característica Principal:**
  - Utiliza uma pilha explícita ou recursividade para guiar a busca

Busca em Profundidade – DFS Algoritmo Recursivo (1)

- Recebe um grafo G = (V, A) (representado por listas de adjacências)
- A busca inicia-se em um vértice qualquer
- Busca em profundidade como um método recursivo:
  1. Suponha que a busca atingiu um vértice u.
  2. Escolhe-se um vizinho não visitado v de u para prosseguir a busca
  3. "Recursivamente" a busca em profundidade prossegue a partir de v
  4. Quando esta busca termina, tenta-se prosseguir a busca a partir de outro vizinho de u. Se não for possível, ela retorna (*backtracking*) ao nível anterior da recursão

Busca em Profundidade – DFS
Algoritmo Recursivo (2)

```
   Entrada: Grafo G=(V, A), vértice inicial v
 1 Marque o vértice v como visitado;
 2 enquanto existir w vizinho de v faça
 3 |   se w é marcado como não visitado então
 4 |   |   Visite a aresta {v, w};
 5 |   |   Marque w como visitado;
 6 |   |   BP(G, w);//chamada recursiva da função
 7 |   fim
 8 |   senão
 9 |   |   se {v, w} não foi visitada ainda então
10 |   |   |   Visite {v, w};
11 |   |   fim
12 |   fim
13 fim
```

Busca em Profundidade – DFS Algoritmo com Pilha Explícita

- Outra forma de entender Busca em Profundidade é imaginar que os vértices são armazenados em uma pilha à medida que são visitados
  - Posteriormente: Compare isto com a Busca em Largura onde os vértices são armazenados em uma fila

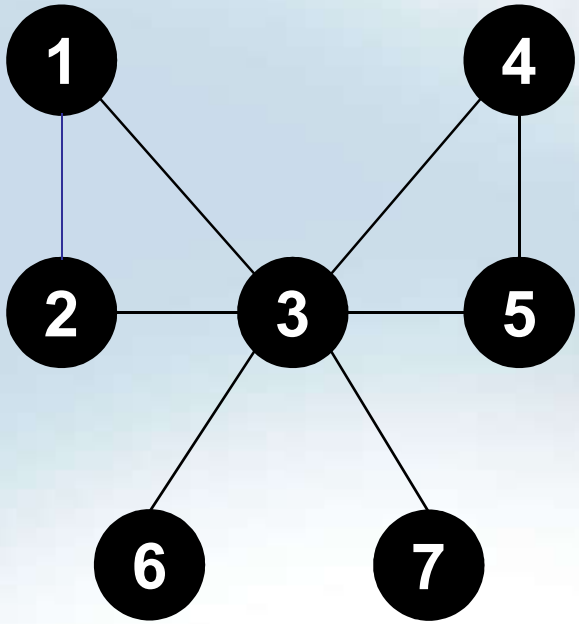
- Busca em profundidade com o uso de uma Pilha Explícita:
  1. Suponha que a busca atingiu um vértice u
  2. Escolhe-se um vizinho não visitado v de u para prosseguir a busca
  3. Empilhe v e repete-se o passo anterior com v
  4. Se nenhum vértice não visitado foi encontrado, então desempilhe um vértice da pilha, digamos u, e volte ao primeiro passo

Busca em Profundidade – DFS Classificação de Arestas

- Ao explorar um grafo G conexo usando a DFS, pode-se categorizar as arestas:
  - **Arestas de Árvore:** Satisfazem ao primeiro **se do algoritmo recursivo (linha 3)** , ou seja, levam à visitação de vértices ainda não visitados
  - **Arestas de Retorno:** Demais arestas
    - Formam ciclos, pois levam a vértices já visitados

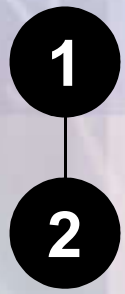
- **Árvore de Profundidade:**
  - A subárvore de G formada pelas arestas de árvore é chamada de Árvore de Profundidade de G

Busca em Profundidade – DFS
Exemplo (1)



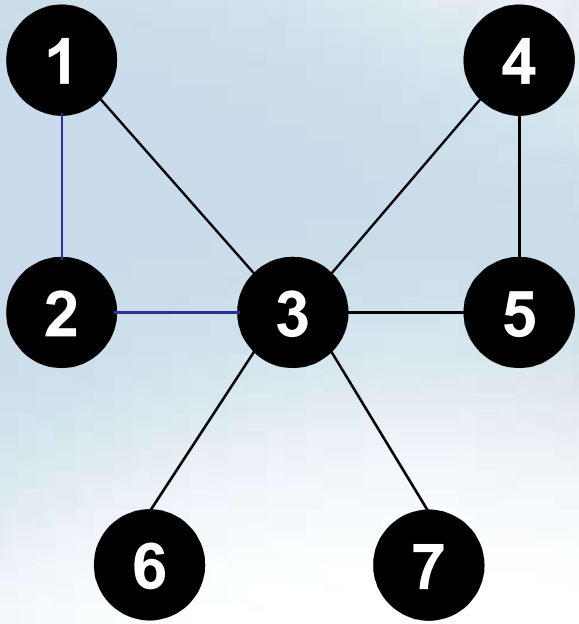
Grafo Não Ordenado

Busca em Profundidade (DFS)



Resultado da Busca - Arestas:
(1, 2)

Busca em Profundidade – DFS
Exemplo (2)

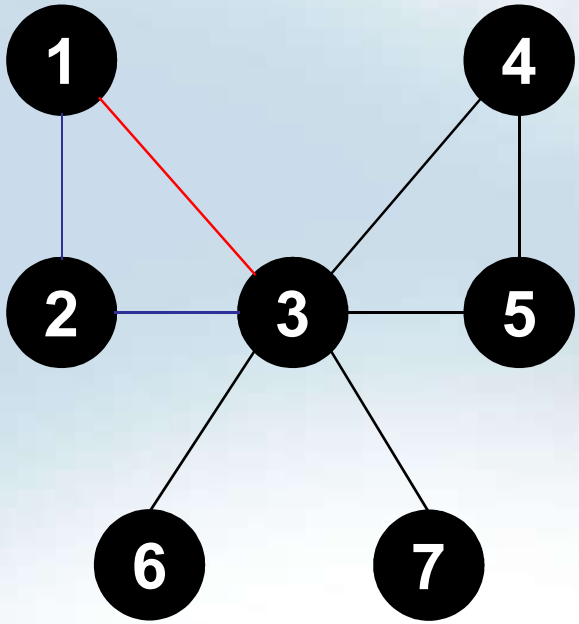


Grafo Não Ordenado

Busca em Profundidade (DFS)

Resultado da Busca - Arestas:
(1, 2) - (2, 3)

Busca em Profundidade – DFS
Exemplo (3)



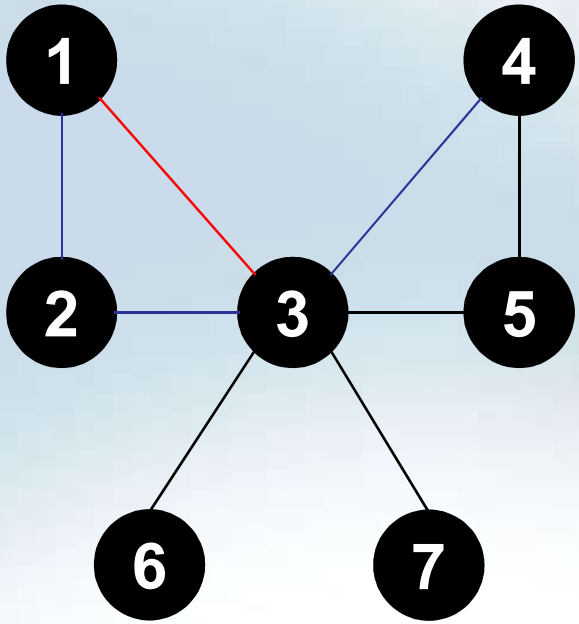
Grafo Não Ordenado

Busca em Profundidade (DFS)

Resultado da Busca em Profundidade:

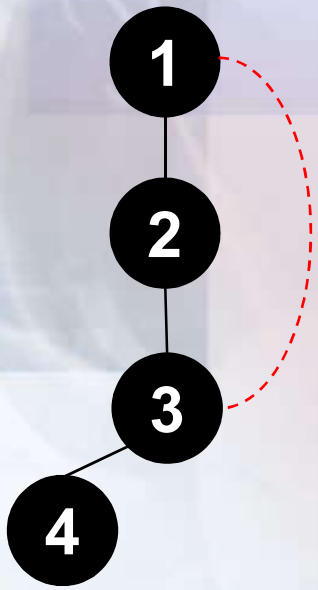
(1, 2) – (2, 3) – (3, 1) –

Busca em Profundidade – DFS
Exemplo (4)



Grafo Não Ordenado

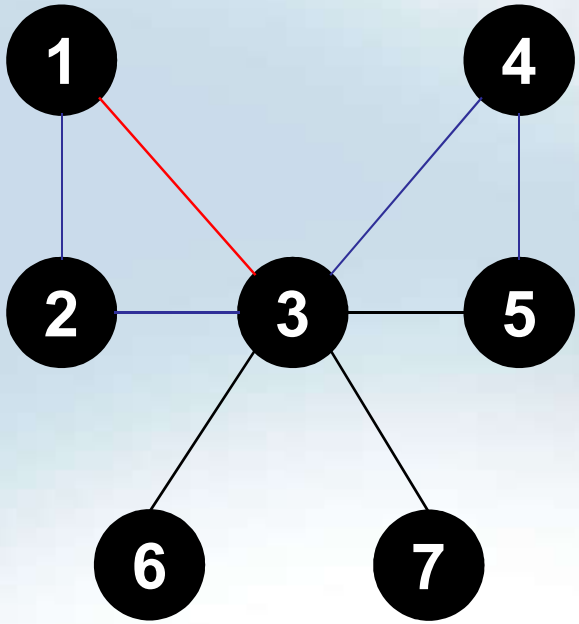
Busca em Profundidade (DFS)



Resultado da Busca em Profundidade:

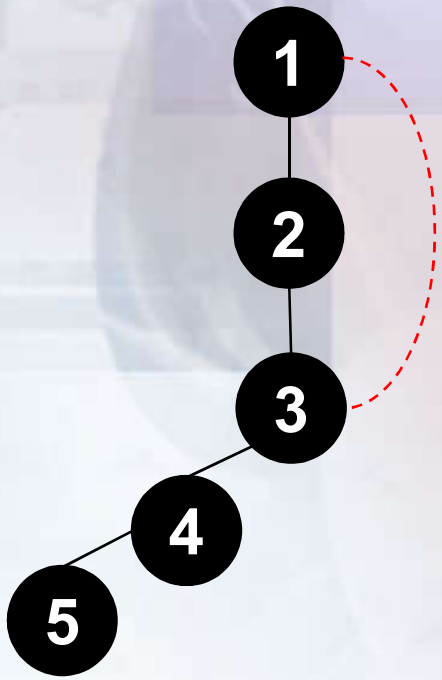
(1, 2) – (2, 3) – (3, 1) – (3, 4)

Busca em Profundidade – DFS
Exemplo (5)



Grafo Não Ordenado

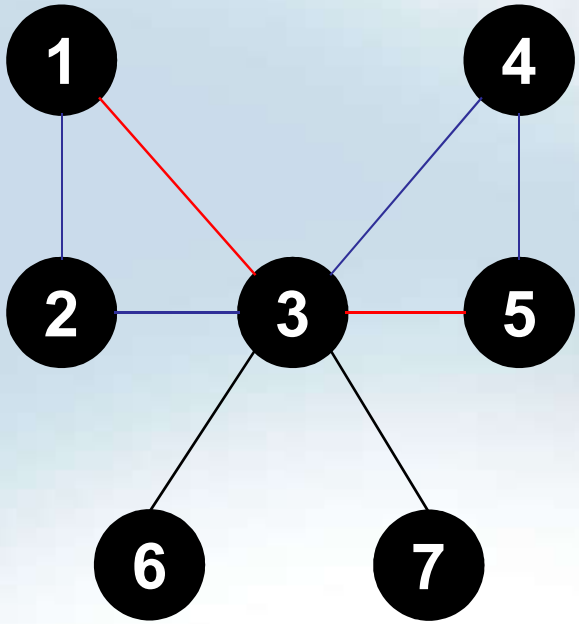
Busca em Profundidade (DFS)



Resultado da Busca em Profundidade:

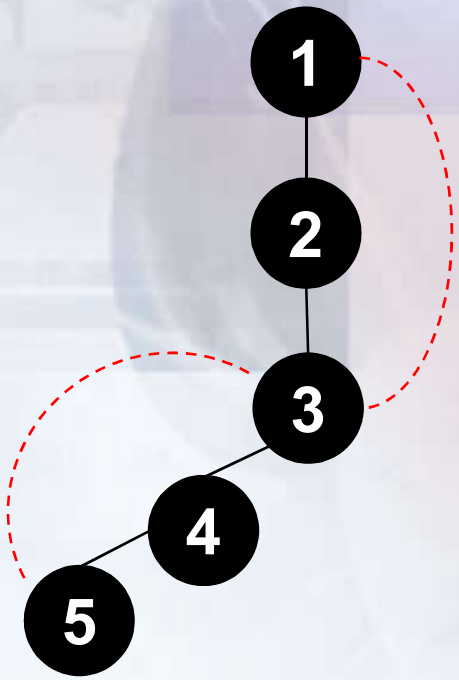
(1, 2) – (2, 3) – (3, 1) – (3, 4) – (4, 5)

Busca em Profundidade – DFS
Exemplo (6)



Grafo Não Ordenado

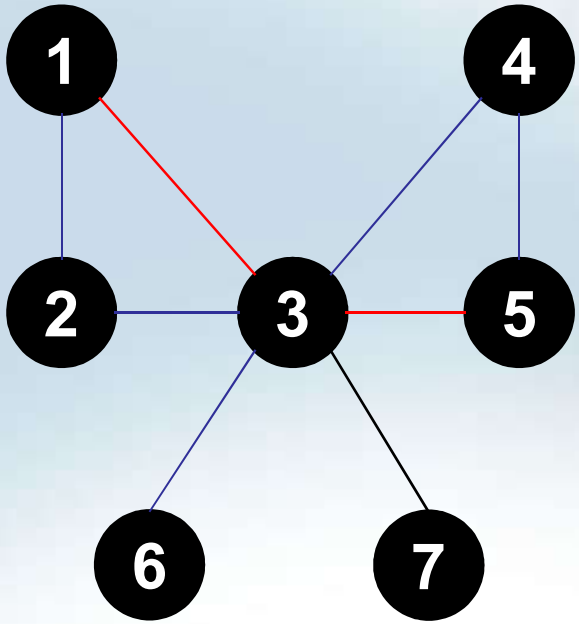
Busca em Profundidade (DFS)



Resultado da Busca em Profundidade:

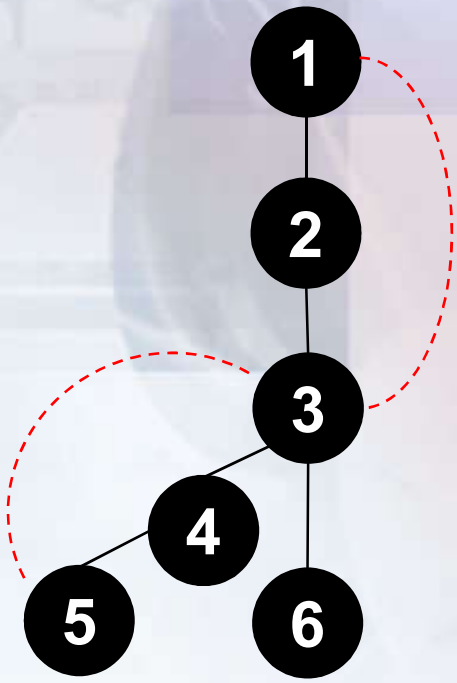
(1, 2) – (2, 3) – (3, 1) – (3, 4) – (4, 5) – (5, 3)

Busca em Profundidade – DFS
Exemplo (7)



Grafo Não Ordenado

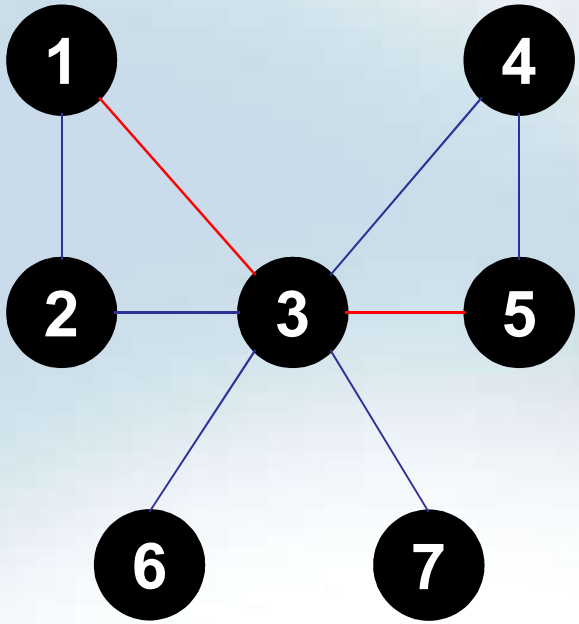
Busca em Profundidade (DFS)



Resultado da Busca em Profundidade:

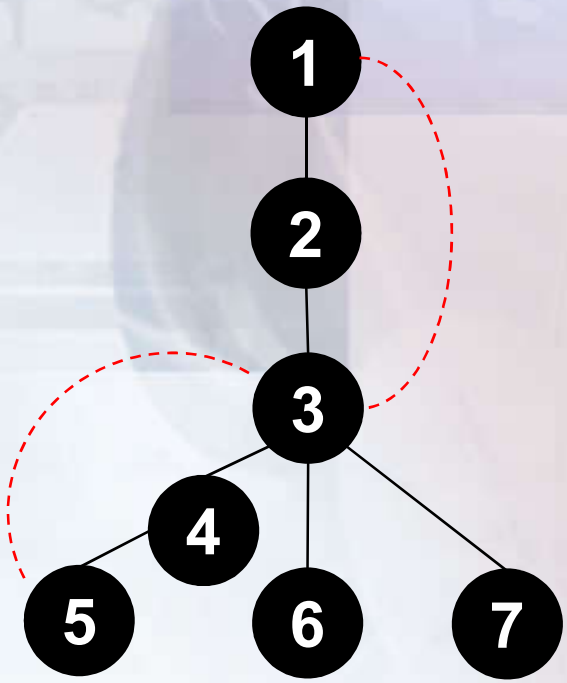
(1, 2) – (2, 3) – (3, 1) – (3, 4) – (4, 5) – (5, 3) – (3, 6)

Busca em Profundidade – DFS
Exemplo (8)



Grafo Não Ordenado

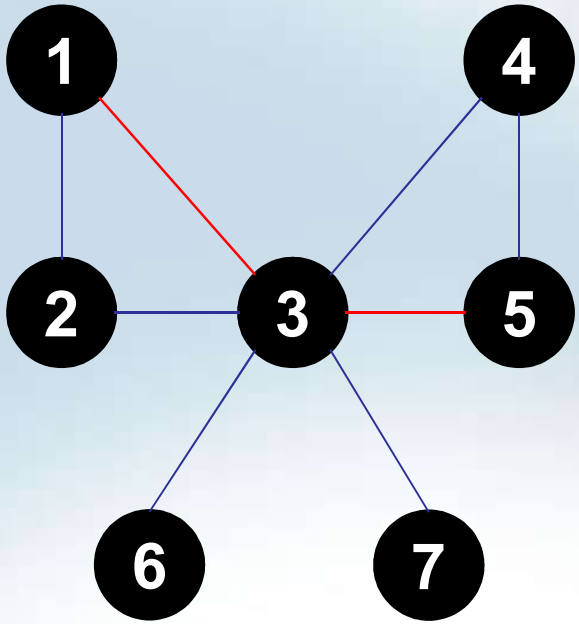
Busca em Profundidade (DFS)



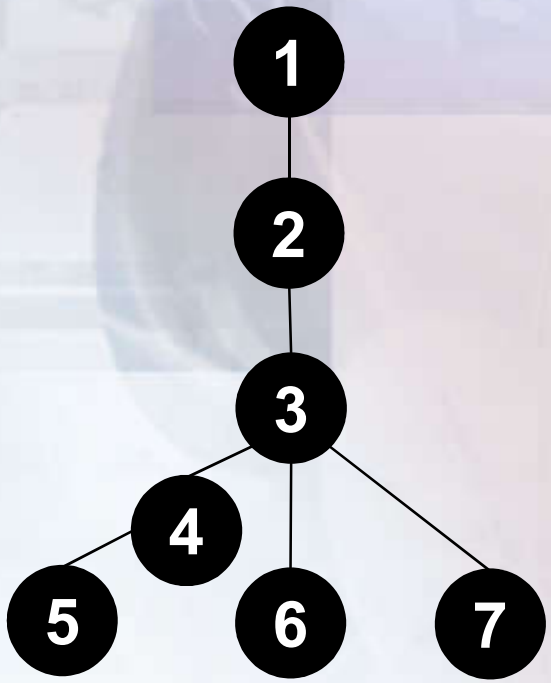
Resultado da Busca em Profundidade:

(1, 2) – (2, 3) – (3, 1) – (3, 4) – (4, 5) – (5, 3) – (3, 6) – (3, 7)

Busca em Profundidade – DFS
Exemplo (9)



Grafo Não Ordenado



Árvore de Profundidade

Busca em Profundidade – DFS Complexidade

- Para cada vértice do grafo, a DFS percorre todos os seus vizinhos. Cada aresta é visitada duas vezes

- Se representarmos o grafo por uma lista de adjacências, a DFS tem complexidade **O(V + A)**

Busca em Profundidade – DFS
Grafos Ordenados (1)

- A aplicação da DFS em grafos direcionados é essencialmente igual à aplicação em grafos não direcionados

- No entanto, mesmo o grafo ordenado sendo conexo, a DFS pode precisar ser chamada repetidas vezes enquanto houver vértices não visitados, retornando uma floresta

- Este é o mesmo caso quando a DFS é aplicada a um Grafo Não Ordenado Desconexo

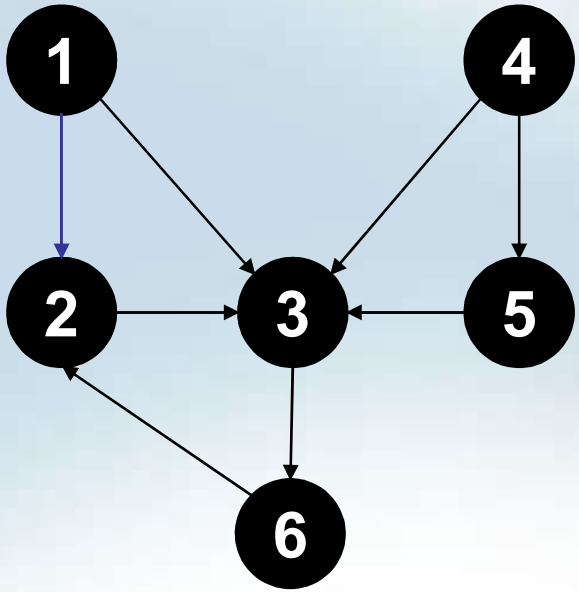
```
   Entrada: Grafo G=(V, A)
1  enquanto existir v ∈ V não visitado faça
2  |   BP(G, v);
3  fim
```

Busca em Profundidade – DFS
Grafos Ordenados (2)

- Ao explorar um grafo G ordenado usando a DFS, pode-se categorizar as arestas

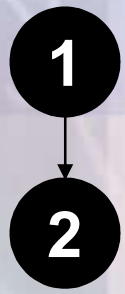
- Sejam o vértice v a origem da aresta e o vértice w o destino da mesma:
  - **Arcos de Avanço:** Caso w seja descendente de v na floresta
  - **Arcos de Retorno:** Caso v seja descendente de w na floresta
  - **Arcos de Cruzamento:** Caso w não seja descendente de v e v não seja descendente de w

Busca em Profundidade – DFS
Exemplo – Grafo Ordenado (1)



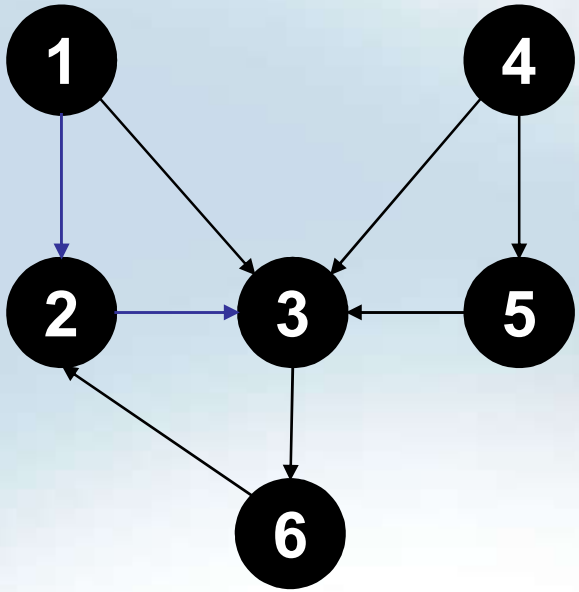
Grafo Ordenado

Busca em Profundidade (DFS)



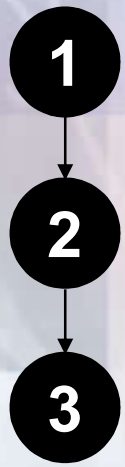
Resultado da Busca - Arestas:
(1, 2)

Busca em Profundidade – DFS
Exemplo – Grafo Ordenado (2)



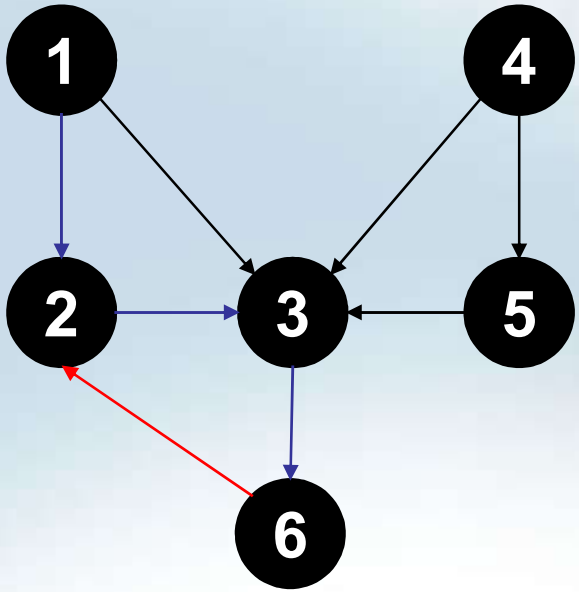
Grafo Ordenado

Busca em Profundidade (DFS)



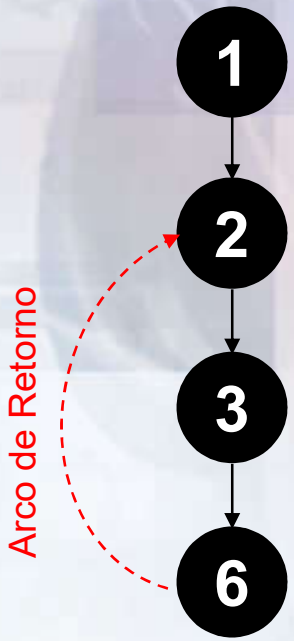
Resultado da Busca - Arestas:
(1, 2) – (2, 3)

Busca em Profundidade – DFS
Exemplo – Grafo Ordenado (3)



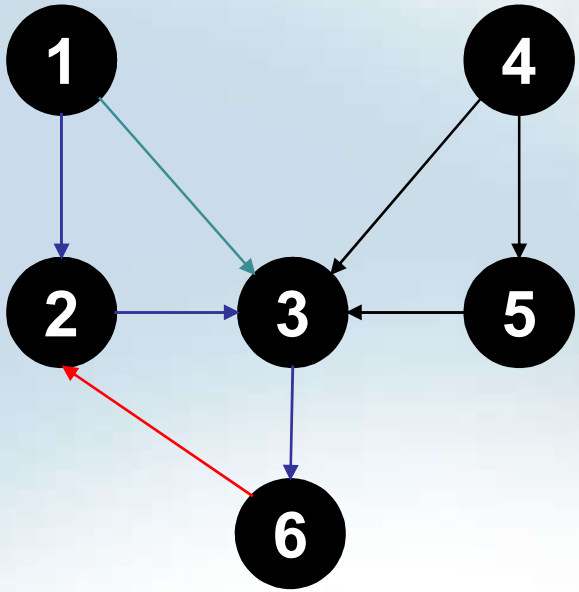
Grafo Ordenado

Busca em Profundidade (DFS)



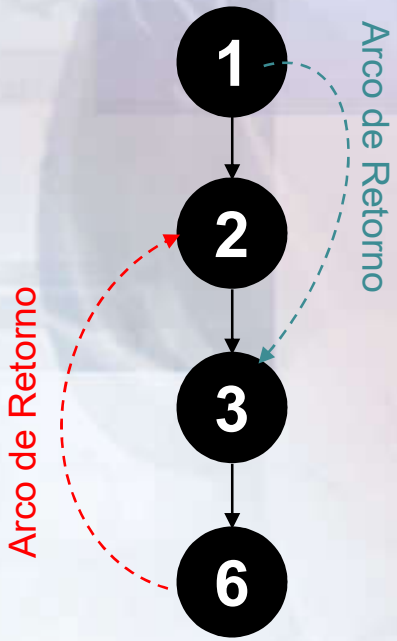
Resultado da Busca - Arestas:
(1, 2) – (2, 3) – (2, 6) – (6, 2)

Busca em Profundidade – DFS
Exemplo – Grafo Ordenado (4)



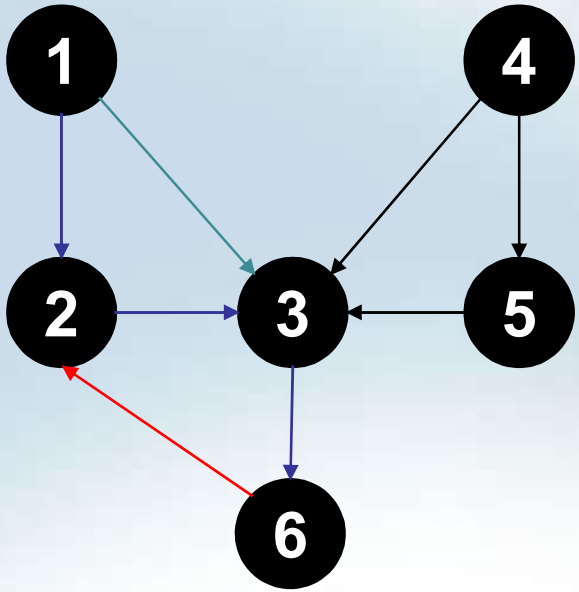
Grafo Ordenado

Busca em Profundidade (DFS)



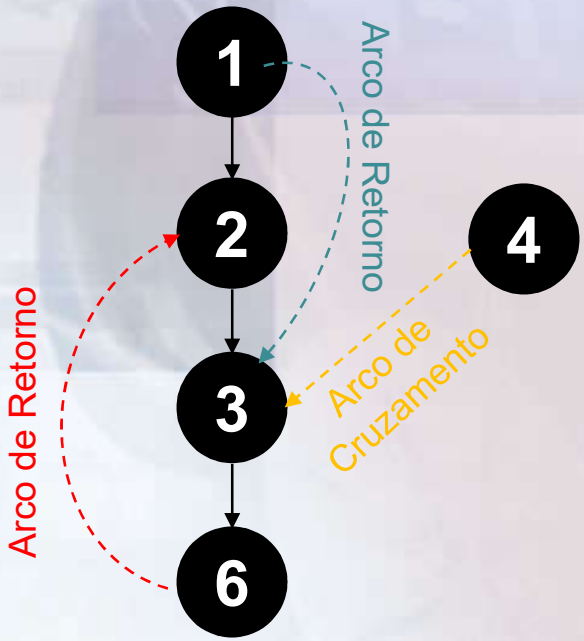
Resultado da Busca - Arestas:
(1, 2) – (2, 3) – (2, 6) – (6, 2) – (1, 3)

Busca em Profundidade – DFS
Exemplo – Grafo Ordenado (5)



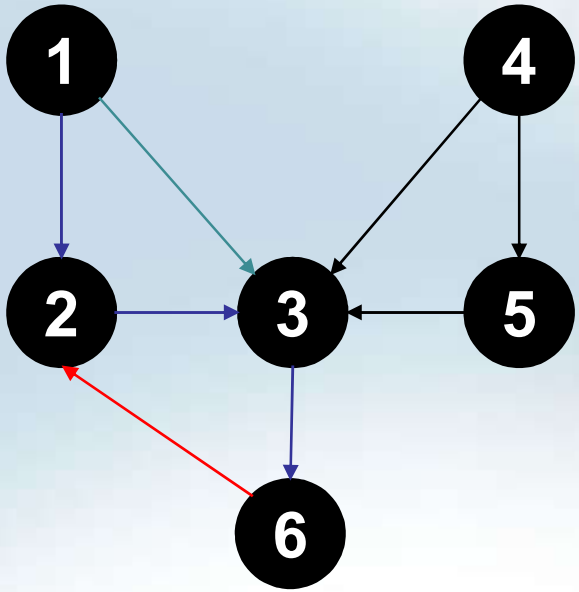
Grafo Ordenado

Busca em Profundidade (DFS)



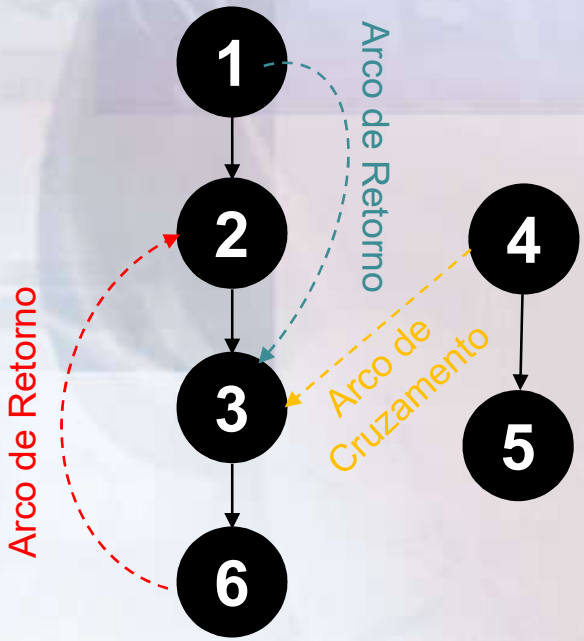
Resultado da Busca - Arestas:
(1, 2) – (2, 3) – (2, 6) – (6, 2) – (1, 3) – (4, 3)

Busca em Profundidade – DFS
Exemplo – Grafo Ordenado (6)



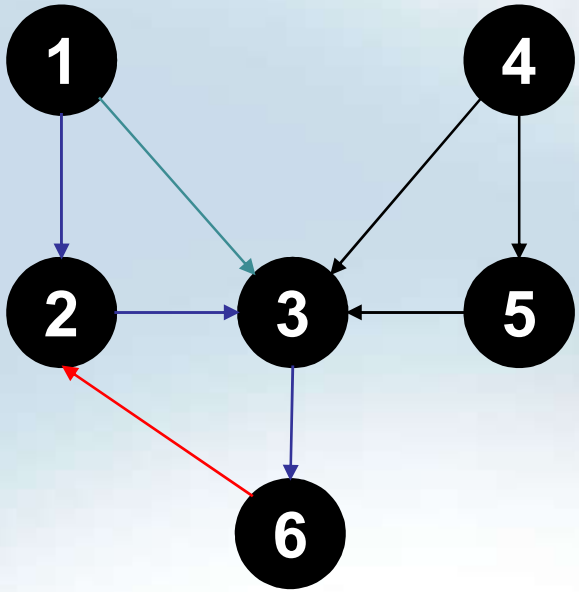
Grafo Ordenado

Busca em Profundidade (DFS)



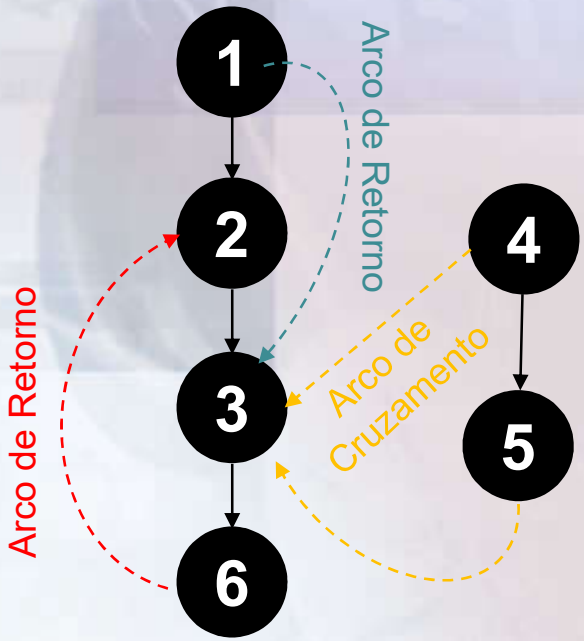
Resultado da Busca - Arestas:
(1, 2) – (2, 3) – (2, 6) – (6, 2) –
(1, 3) – (4, 3) – (4, 5)

Busca em Profundidade – DFS
Exemplo – Grafo Ordenado (7)



Grafo Ordenado

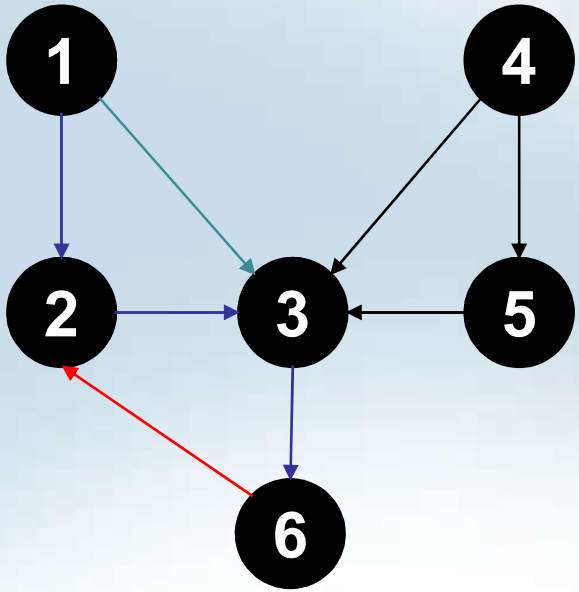
Busca em Profundidade (DFS)



Resultado da Busca - Arestas:

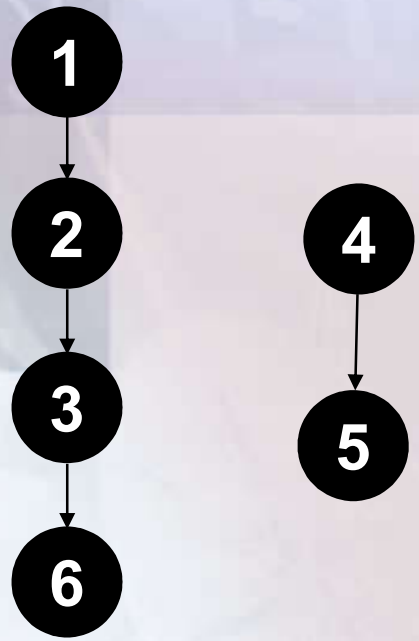
(1, 2) – (2, 3) – (2, 6) – (6, 2) –
(1, 3) – (4, 3) – (4, 5) – (5, 3)

Busca em Profundidade – DFS
Exemplo – Grafo Ordenado (8)



Grafo Ordenado

Busca em
Profundidade (DFS)



Floresta de Profundidade

Busca em Largura – BFS

- A **Busca em Largura visita** todos **os vértices** de um grafo, usando como **critério o vértice visitado menos recentemente** e **cuja vizinhança ainda não foi explorada**

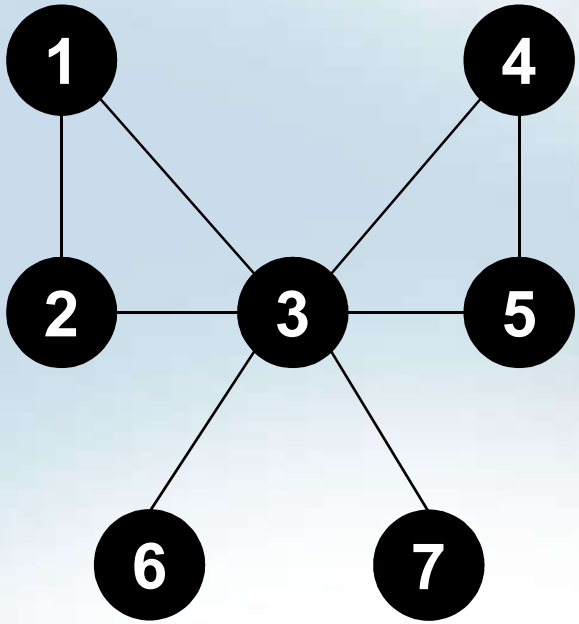
- **Característica Principal**: utiliza uma fila guiar a busca

- **Atuação em camadas**:
  - Inicialmente são considerados os vértices com distância 0 do vértice inicial
  - Na iteração 1 são visitados os vértices com distância 1; prosseguindo, de modo genérico, na iteração d será adicionada uma camada com todos os vértices com distância d do vértice inicial
  - Cada novo vértice visitado é adicionado no final de uma fila Q
  - Cada vértice da fila é removido depois que toda a vizinhança for visitada
  - A busca termina quando a fila se torna vazia

Busca em Largura – BFS
Algoritmo

```
   Entrada: Grafo G=(V, A), vértice inicial v
1  Crie uma fila Q vazia;
2  Marque v como visitado;
3  Insira v em Q;
4  enquanto Q ≠ ∅ faça
5      v ← remove elemento de Q;
6      para todo vértice w vizinho de v faça
7          se w é marcado como não visitado então
8              Visite a aresta {v, w};
9              Insira w em Q;
10             Marque w como visitado;
11         fim
12         senão
13             se {v, w} não foi visitada ainda então
14                 Visite {v, w};
15             fim
16         fim
17     fim
18 fim
```

Busca em Largura – BFS
Exemplo (1)

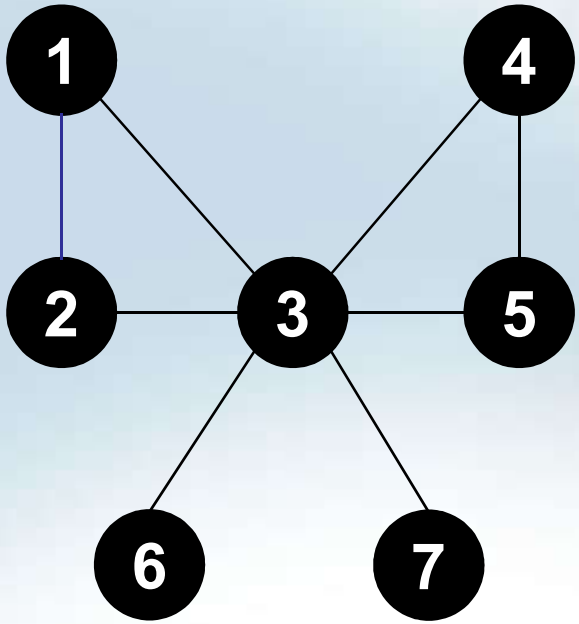


Grafo Não Ordenado

Busca em Largura (BFS)

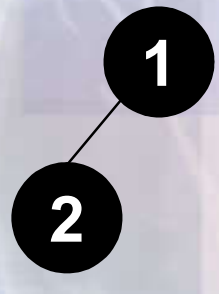
Resultado da Busca:
Q (Fila) = {1}
Arestas: -

Busca em Largura – BFS
Exemplo (2)



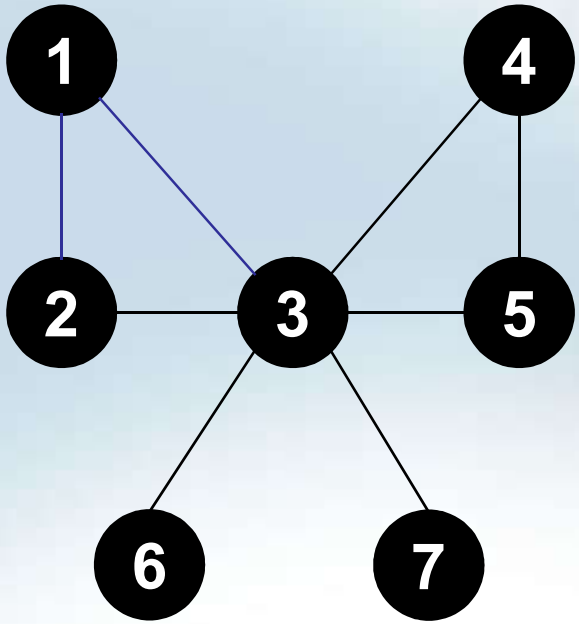
Grafo Não Ordenado

Busca em Largura (BFS)



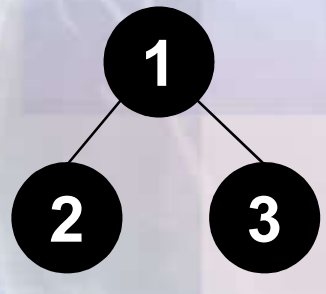
Resultado da Busca:
Q (Fila) = {2}
Arestas: (1, 2)

Busca em Largura – BFS
Exemplo (3)



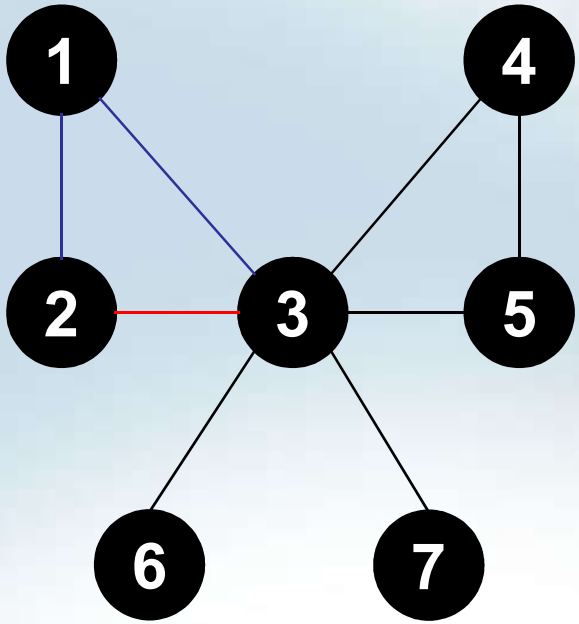
Grafo Não Ordenado

Busca em Largura (BFS)



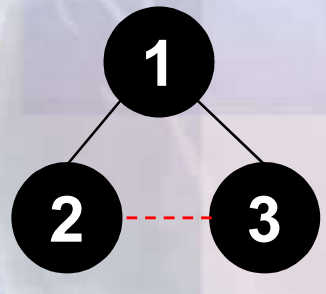
Resultado da Busca:
Q (Fila) = {2, 3}
Arestas: (1, 2) – (1, 3)

Busca em Largura – BFS
Exemplo (4)



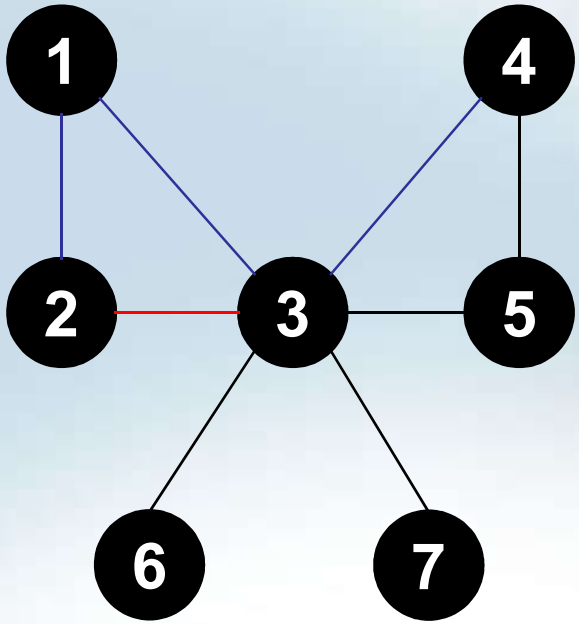
Grafo Não Ordenado

Busca em Largura (BFS)



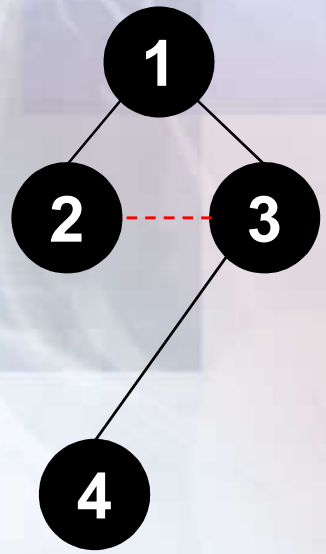
Resultado da Busca:
Q (Fila) = {3}
Arestas: (1, 2) – (1, 3) – (2, 3)

Busca em Largura – BFS
Exemplo (5)



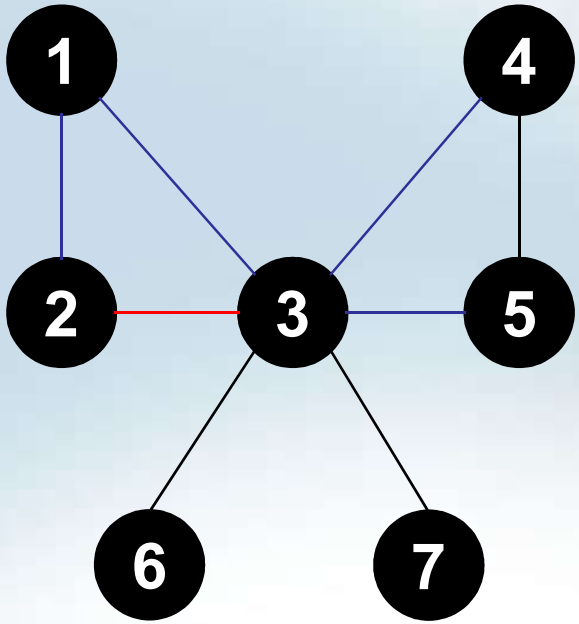
Grafo Não Ordenado

Busca em Largura (BFS)



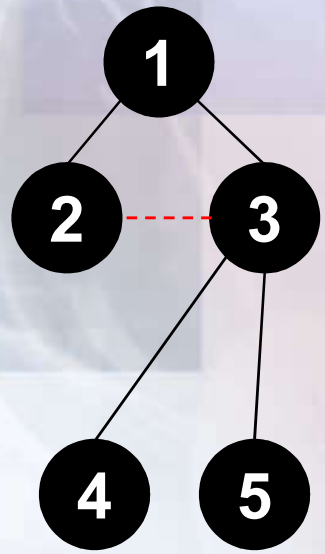
Resultado da Busca:
Q (Fila) = {3, 4}
Arestas: (1, 2) – (1, 3) – (2, 3) – (3, 4)

Busca em Largura – BFS
Exemplo (6)



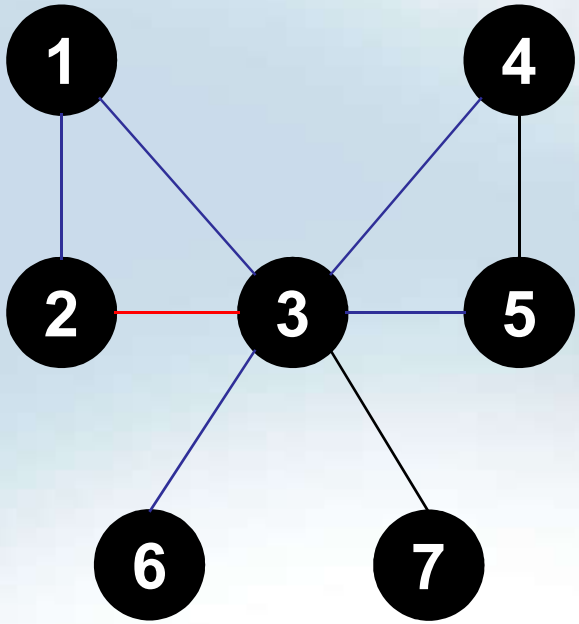
Grafo Não Ordenado

Busca em Largura (BFS)



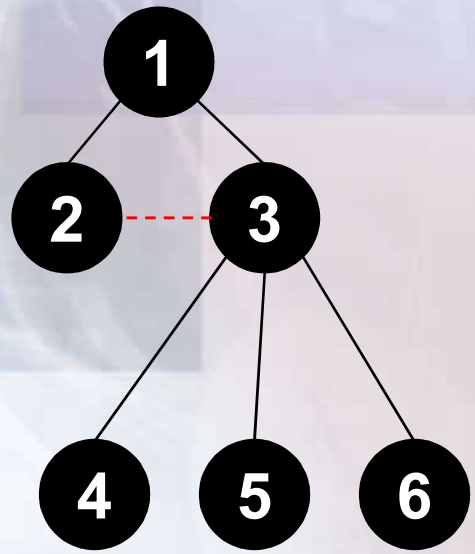
Resultado da Busca:
Q (Fila) = {4, 5}
Arestas: (1, 2) – (1, 3) – (2, 3) – (3, 4) – (3, 5)

Busca em Largura – BFS
Exemplo (7)



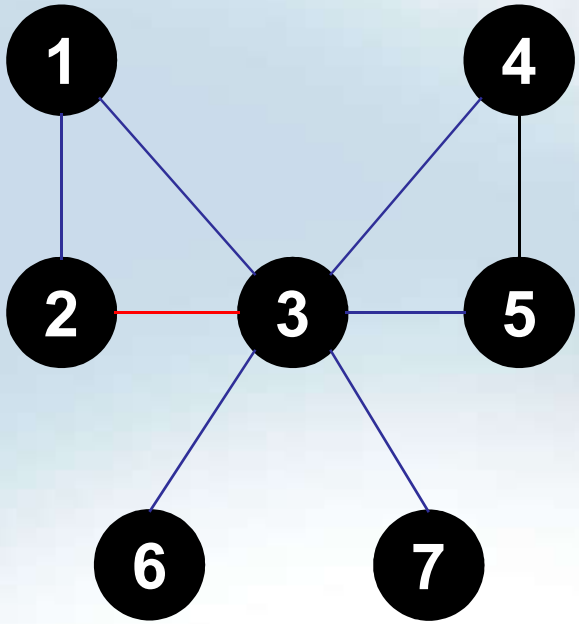
Grafo Não Ordenado

Busca em Largura (BFS)



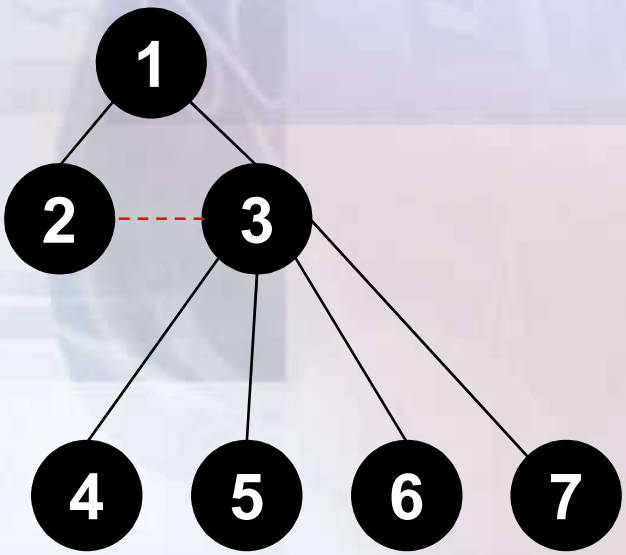
Resultado da Busca:
Q (Fila) = {4, 5, 6}
Arestas: (1, 2) – (1, 3) – (2, 3) – (3, 4) – (3, 5) – (3, 6)

Busca em Largura – BFS
Exemplo (8)



Grafo Não Ordenado

Busca em Largura (BFS)

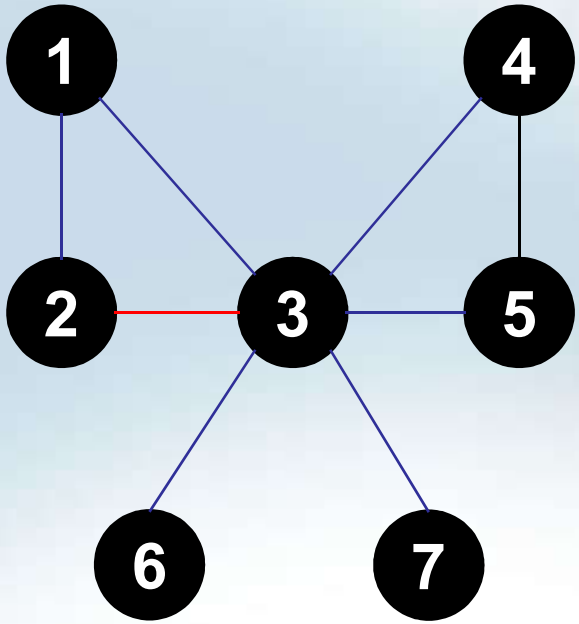


Resultado da Busca:

Q (Fila) = {4, 5, 6, 7}

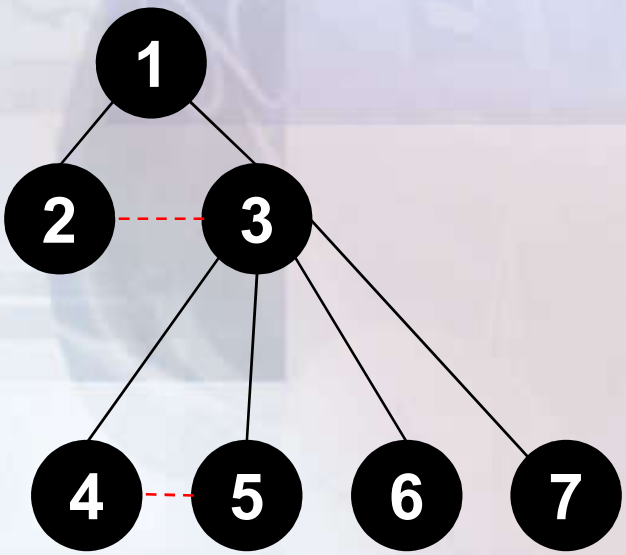
Arestas: (1, 2) – (1, 3) – (2, 3) – (3, 4) – (3, 5) – (3, 6) – (3, 7)

Busca em Largura – BFS
Exemplo (9)



Grafo Não Ordenado

Busca em Largura (BFS)

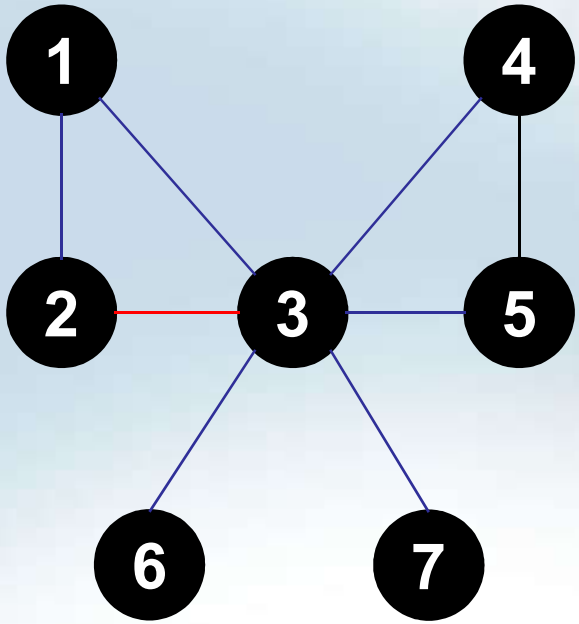


Resultado da Busca:

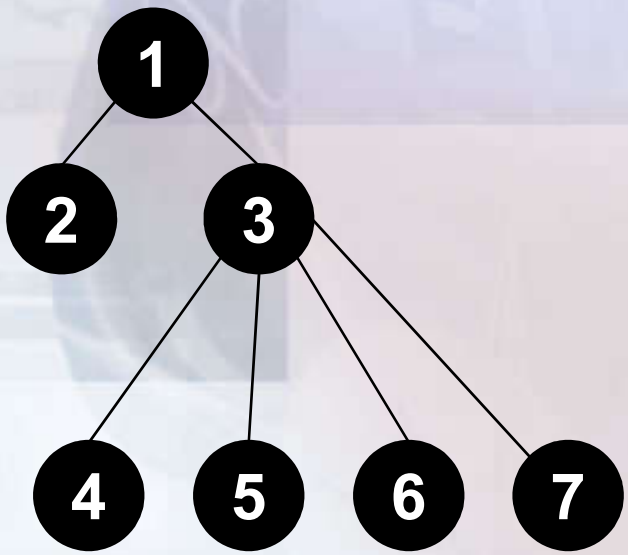
Q (Fila) = {5, 6, 7}

Arestas: (1, 2) – (1, 3) – (2, 3) – (3, 4) – (3, 5) – (3, 6) – (3, 7) – (4, 5)

Busca em Largura – BFS
Exemplo (10)



Grafo Não Ordenado



Árvore de Largura

Busca em Largura– BFS Complexidade

- Cada vértice só entra na fila uma vez
- Inserir e remover na fila possuem complexidade constante, realizadas |V| vezes cada
- A lista de adjacências de cada vértice é examinada apenas uma vez, e a soma dos comprimentos de todas as listas é Θ(A)
- Logo, se representarmos o grafo por uma lista de adjacências, a BFS tem **complexidade O(V + A)**

DFS vs. BFS

- **DFS - Busca em Profundidade:**
  - Incursões profundas no grafo, voltando somente quando não existem mais vértices desconhecidos pela frente
  - Marca o vértice antes de visitar toda sua vizinhança
  - Uso de pilha
- **BFS - Busca em Largura:**
  - Busca progride em “largura”: certifica-se de que vizinhos próximos sejam visitados primeiro a partir de v;
  - Marca o vértice depois de visitar toda sua vizinhança
  - Uso de fila
- **Visualização dos algoritmos**: https://visualgo.net/

PANC: Projeto e Análise de Algoritmos

Aula 13: Grafos

Breno Lisi Romano

Dúvidas???

http://sites.google.com/site/blromano

Instituto Federal de São Paulo – IFSP São João da Boa Vista
Bacharelado em Ciência da Computação – 3º Semestre

INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SÃO PAULO
Campus São João da Boa Vista