
Grafos: Busca em Largura

SCC0216/503 Modelagem Computacional em Grafos

Thiago A. S. Pardo

Maria Cristina F. Oliveira

Percorrendo um grafo

- Percorrer um grafo é um **problema fundamental**
- Deve-se ter uma forma **sistemática** de visitar as arestas e os vértices: objetivo é `processar` todos os vértices
- O algoritmo deve ser suficientemente **flexível** para se adequar à diversidade de grafos

Percorrendo um grafo

- **Eficiência**

- Não deve haver repetições (desnecessárias) de visitas a um vértice e/ou aresta

Percorrendo um grafo

- Correção

- Todos os vértices devem ser visitados e processados

Percorrendo um grafo

■ Solução

- Há duas possibilidades
 - Busca em largura
 - Busca em profundidade
- Marcar os vertices:
 - Ainda não visitados, já visitados, processados
 - Estrutura de dados de apoio para 'registrar' os vértices que ainda falta visitar (e processar)

Percorrendo um grafo

- Solução

- Há duas possibilidades

- **Busca em largura (usando uma fila)**

- Busca em profundidade (usando uma pilha)

BFS (Busca em Largura)

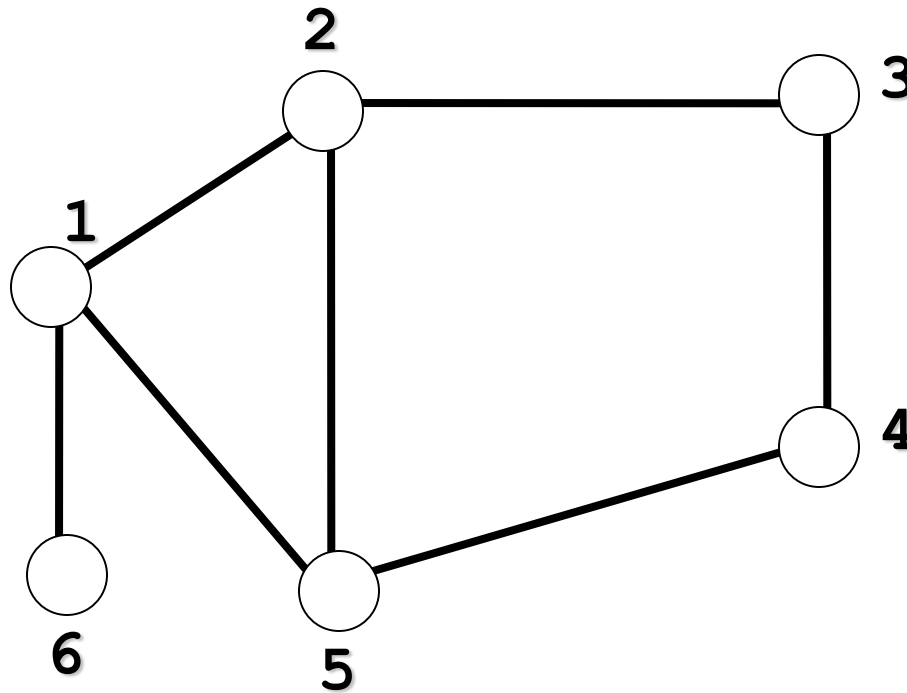
- BFS – *Breadth-First Search*
 - Também chamada de busca em “amplitude”
 - Começa *visitando* um vértice arbitrário v
 - Em seguida, todos os nós a uma distância k dele são *visitados* antes de visitar os vértices a uma distância $k+1$
 - Ao final, `descobre` todos os vértices alcançáveis a partir de v

BFS (Busca em Largura)

- BFS – *Breadth-First Search*
 - É comum utilizar uma codificação de cores para identificar os vértices:
 - ainda não visitados (**branco**),
 - já visitados (**cinza**), e
 - completamente processados (**preto**)

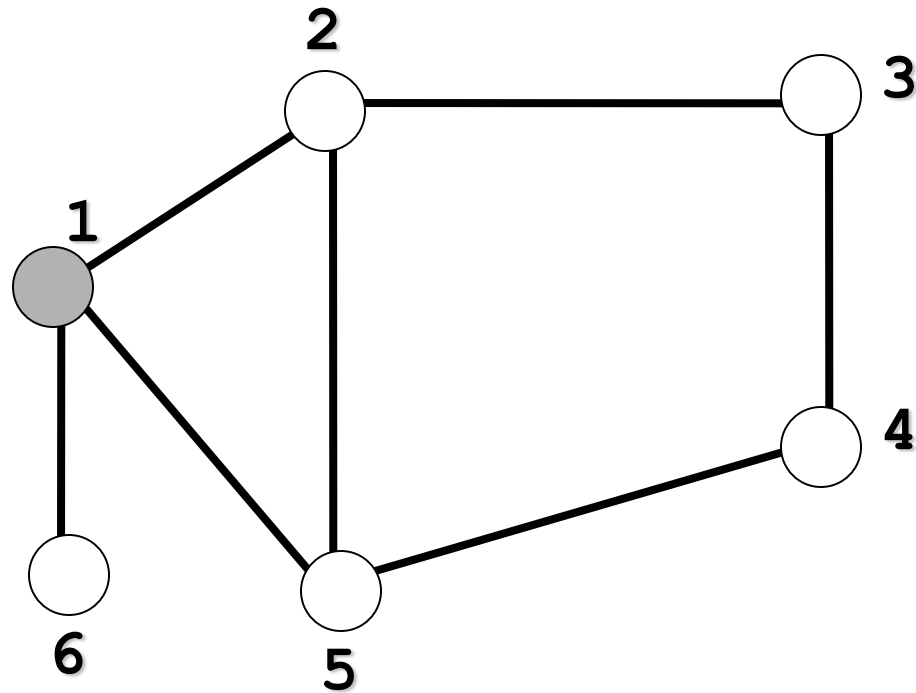
BFS – exemplo

Percorrendo um Grafo: BFS



BFS – exemplo

Percorrendo um Grafo: BFS

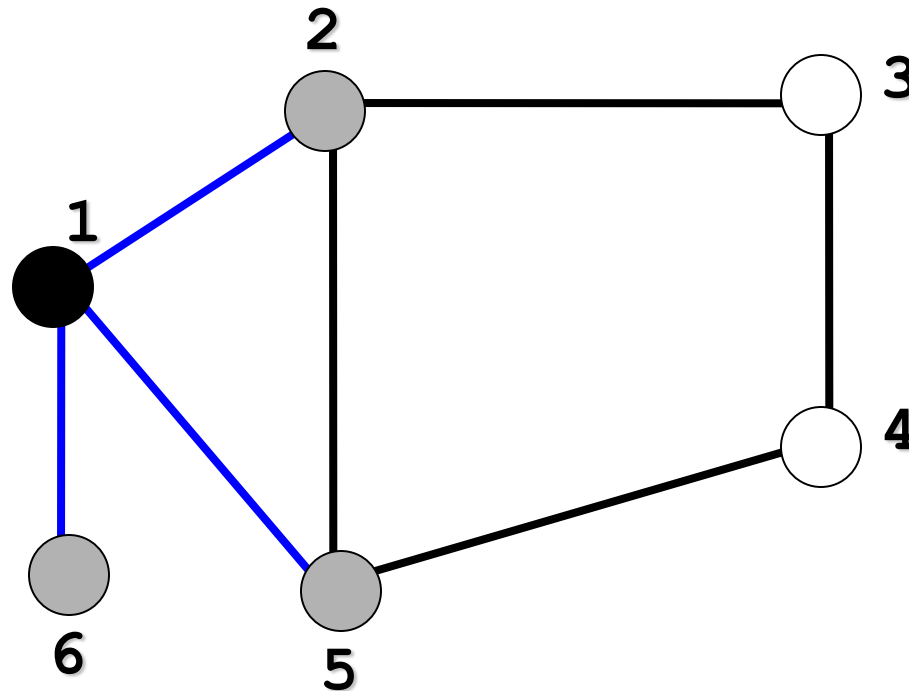


Nó inicial: 1

BFS – exemplo

Percorrendo um Grafo: BFS

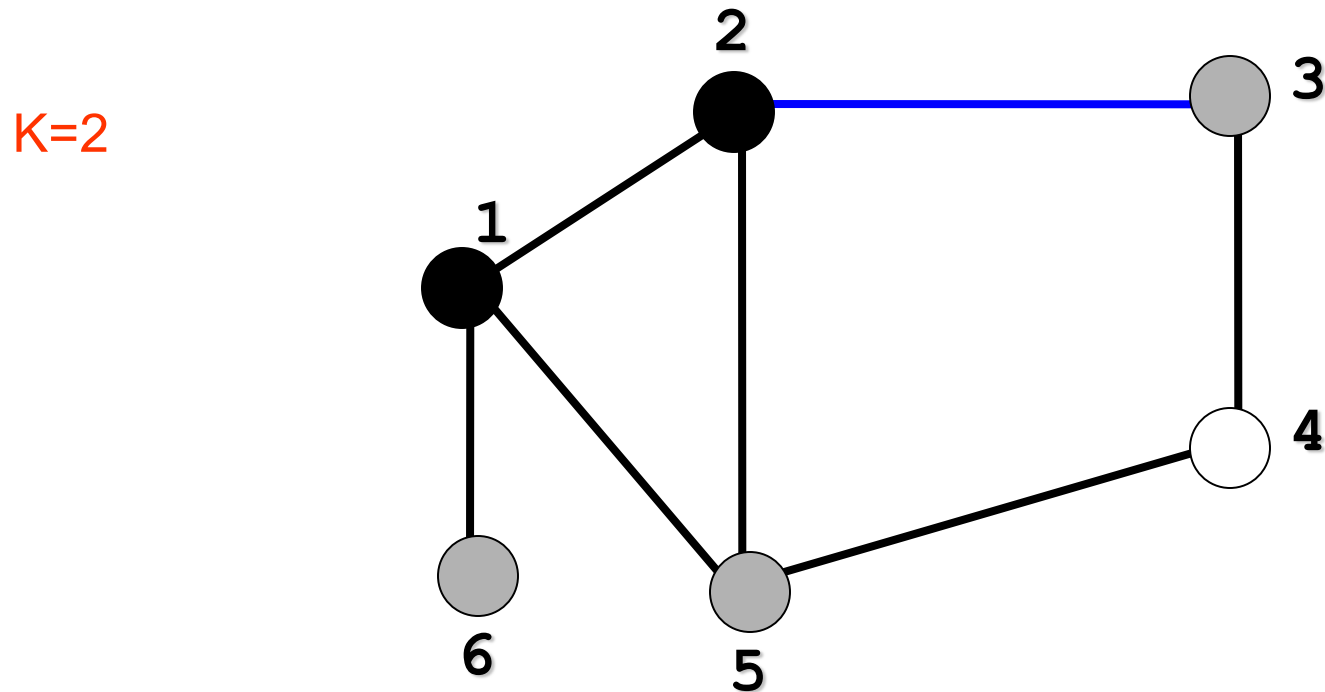
K=1



Visita (e marca) todos os nós não visitados adjacentes a 1: 2, 5 e 6

BFS – exemplo

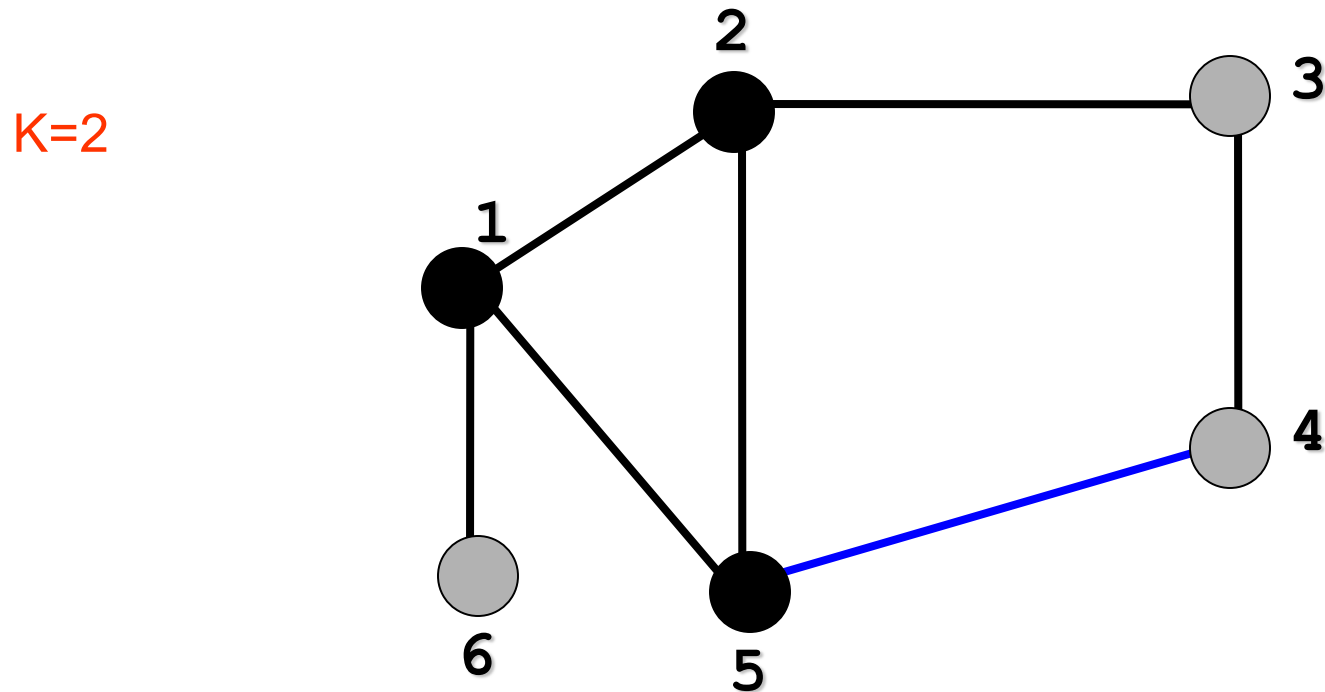
Percorrendo um Grafo: BFS



Visita (e marca) todos os nós não visitados adjacentes a 2: 3

BFS – exemplo

Percorrendo um Grafo: BFS

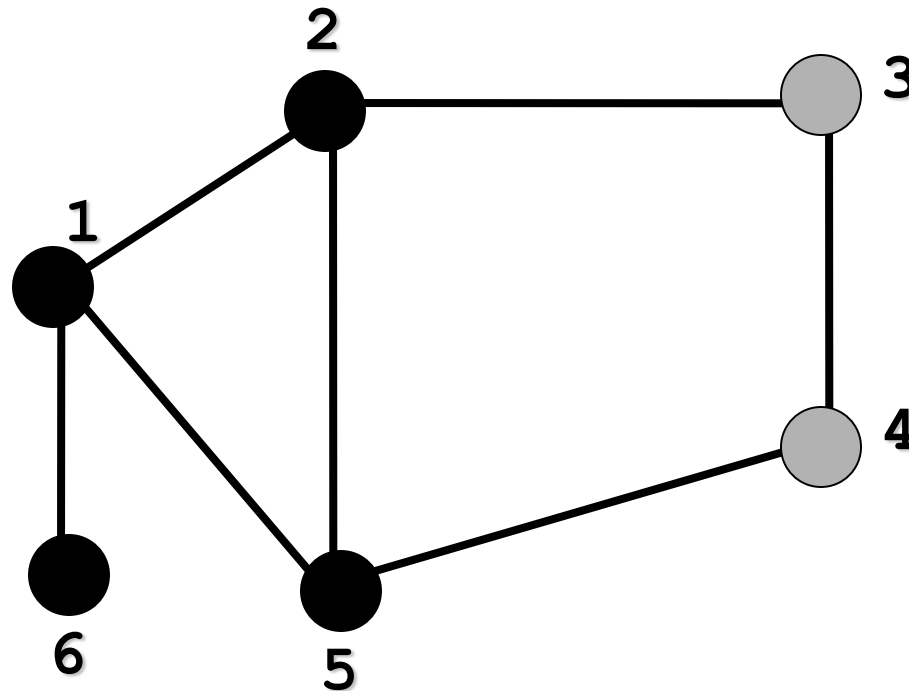


Visita (e marca) todos os nós não visitados adjacentes a 5: 4

BFS – exemplo

Percorrendo um Grafo: BFS

K=2

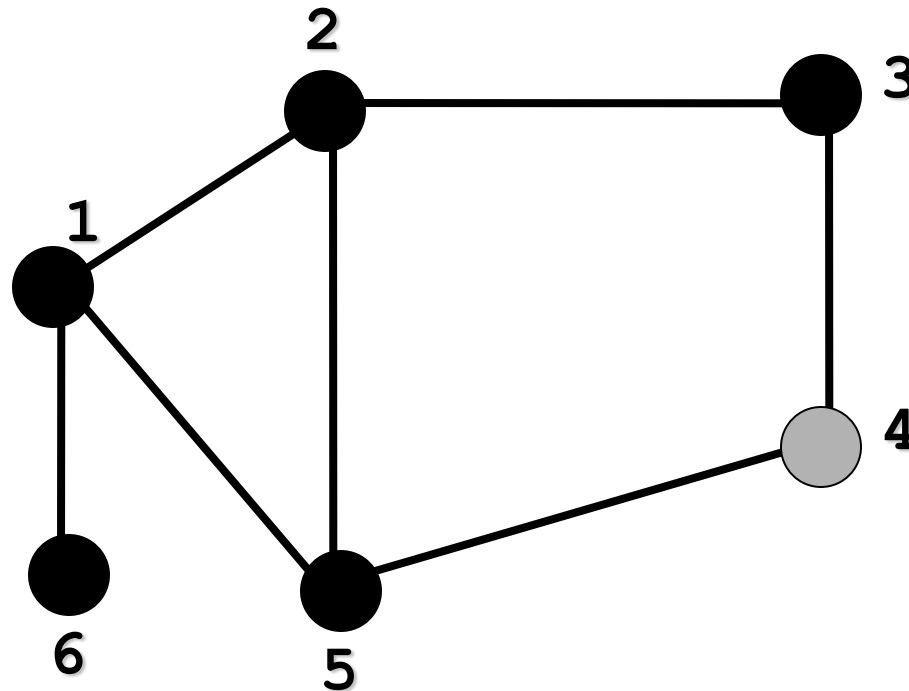


Visita (e marca) todos os nós não visitados adjacentes a 6: nenhum

BFS – exemplo

Percorrendo um Grafo: BFS

K=3

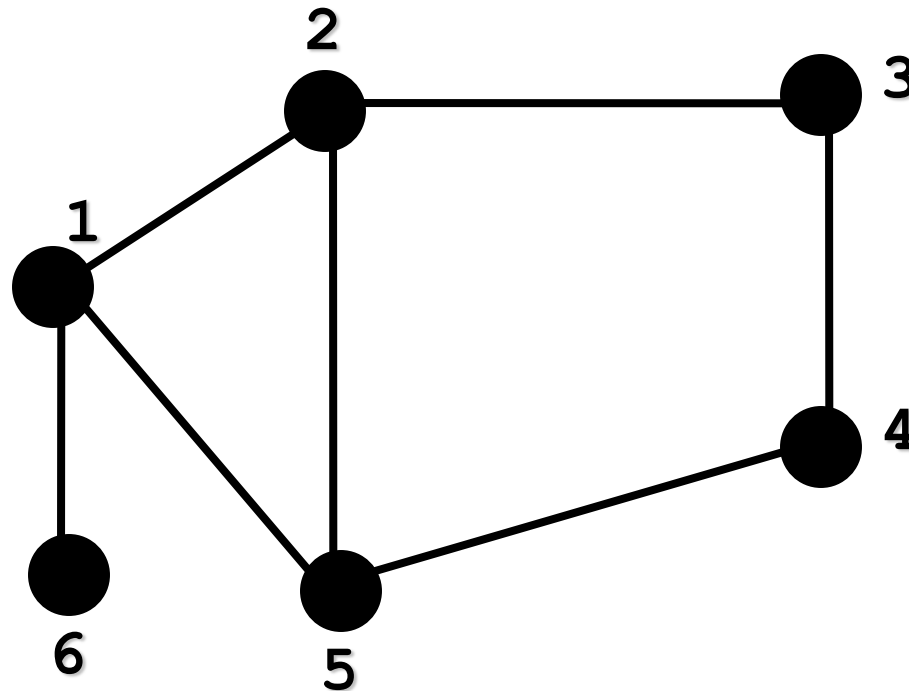


Visita (e marca) todos os nós não visitados adjacentes a 3: nenhum

BFS – exemplo

Percorrendo um Grafo: BFS

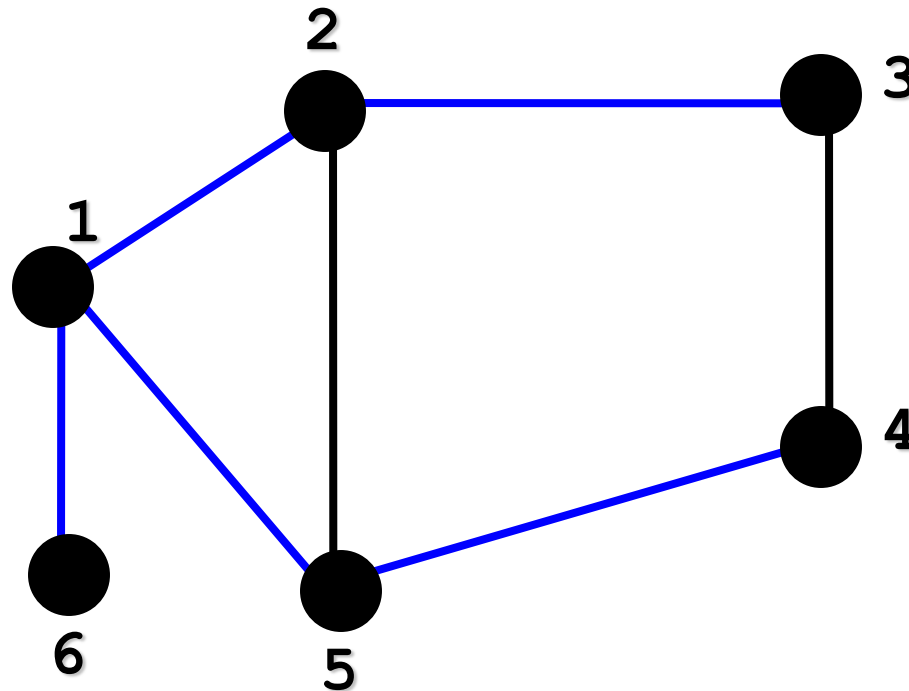
K=3



Visita (e marca) todos os nós não visitados adjacentes a 4: nenhum

BFS – exemplo

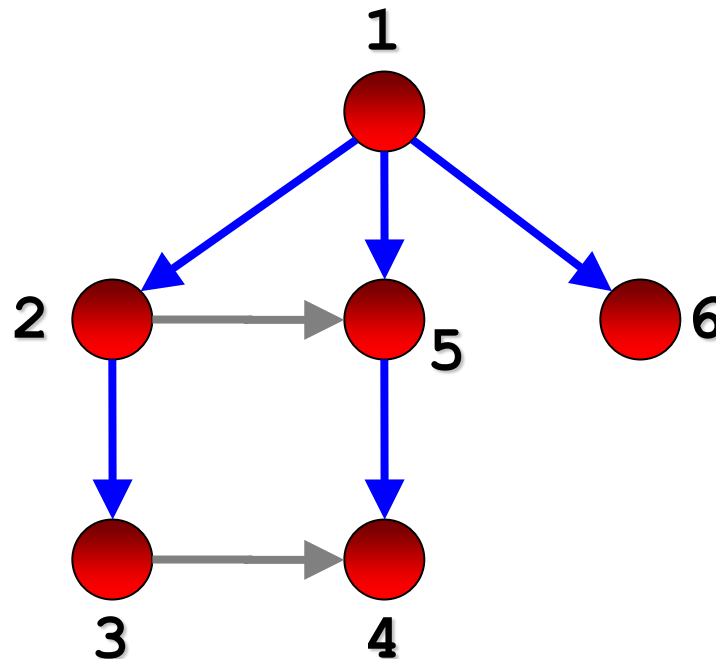
Percorrendo um Grafo: BFS



Em azul as arestas 'percorridas' no processo!

BFS – exemplo

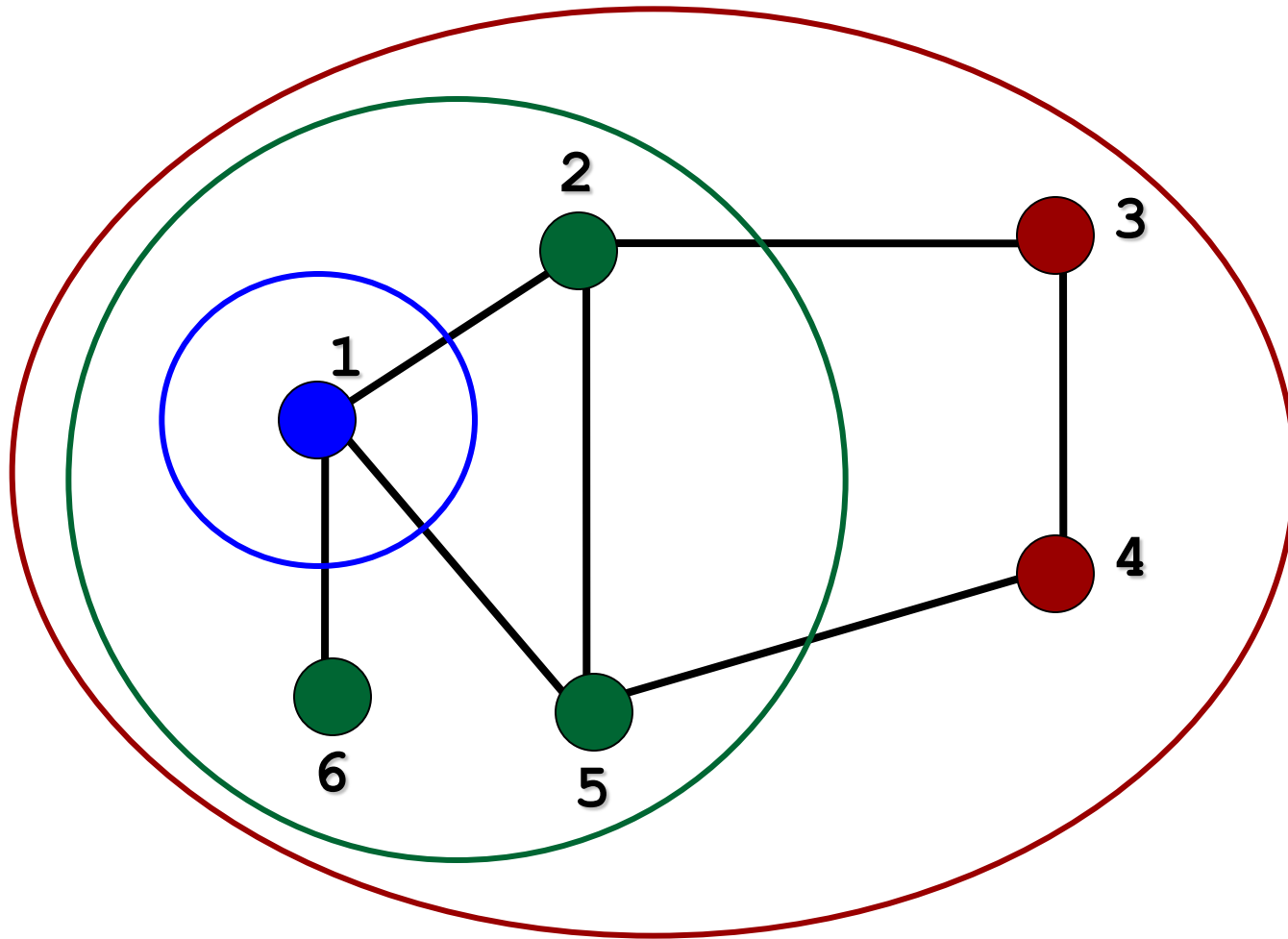
Percorrendo um Grafo: **árvore de busca em largura**



Atenção: as arestas (2,5) e (3,4) não foram percorridas!

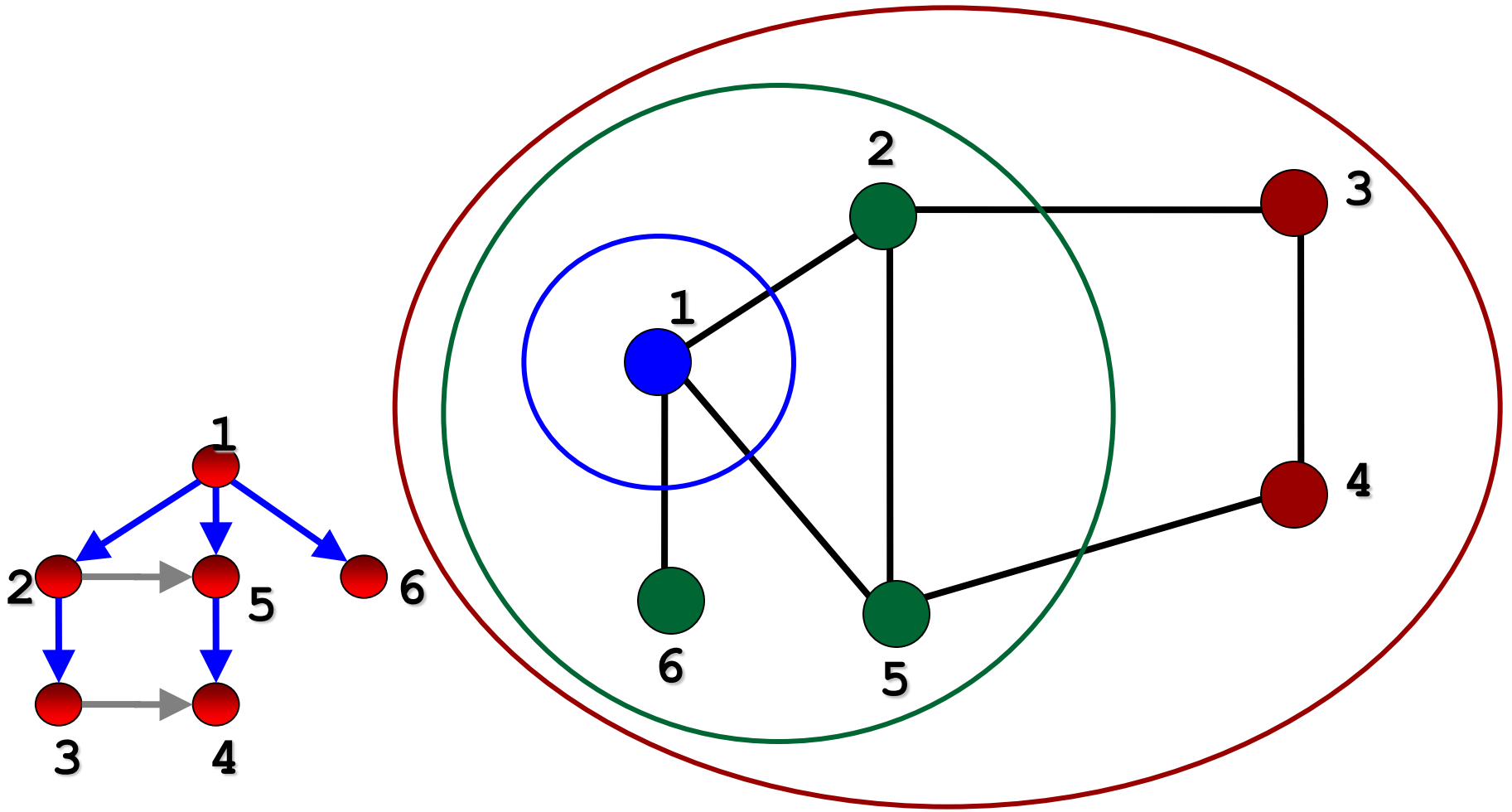
BFS – exemplo

Percorre-se o grafo como se propagasse uma onda na água!



BFS – exemplo

Percorre-se o grafo como se houvesse uma onda na água!



BFS (Busca em Largura)

- BFS – *Breadth-First Search*
 - Todos os vértices são inicializados como **brancos**
 - Quando um vértice v é ‘descoberto’ pela primeira vez, ele se torna **cinza** (visitado)
 - Quando todos os vértices adjacentes a v são descobertos, v torna-se **preto** (processado)

```
1 procedure BFS( $G$ ,  $start\_v$ )
2   Seja  $Q$  uma queue
3   rotule  $start\_v$  como cinza,  $dist = -1$ 
4    $Q.enqueue(start\_v)$ 
5   while  $Q$  não está vazia do
6      $v := Q.dequeue()$ ,  $dist = dist + 1$ 
7     print( $v$ ,  $dist$ )
8     for all arestas  $(v, w)$  in  $G.adjacentEdges(v)$  do
9       if  $w$  não está rotulado como cinza then
10         rotule  $w$  como cinza
11          $Q.enqueue(w)$ 
12     rotule  $v$  com preto
```

Input: Um grafo G , um vértice inicial $start_v$, todos os vértices rotulados como *branco*

Output: A sequência de vértices rotulados como *preto*
(processados na ordem da visita BFS)

BFS (Busca em Largura)

- BFS – *Breadth-First Search*
 - Também é comum a implementação armazenar a **distância** de cada vértice em relação ao vértice no qual se iniciou a busca
 - Útil em aplicações que precisam calcular o **caminho mais curto a partir de um vértice**

BFS (Busca em Largura)

■ Algoritmo

- Usa uma **fila** para organizar quais vértices precisam ser visitados
 1. A fila começa com o vértice inicial
 2. O primeiro vértice da fila é recuperado e processado, e seus vértices adjacentes são inseridos no fim da fila
 3. Se fila vazia, processo é encerrado; senão, retorna ao passo 2

BFS (Busca em Largura)

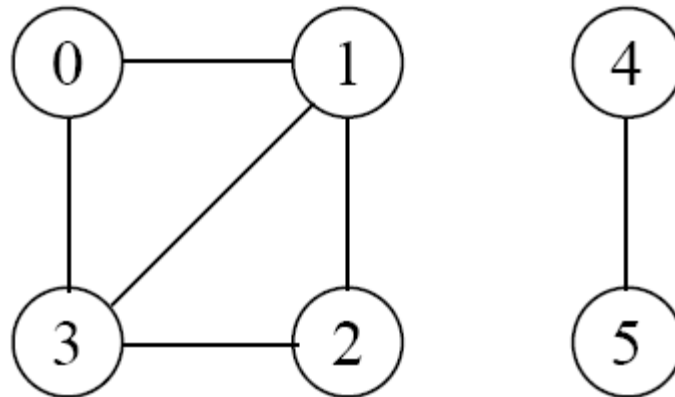
■ Algoritmo

- Ou seja: a fila, por si só, **garante** que os vértices a uma distância k de um vértice v sejam processados antes dos vértices a uma distância $k+1$ de v

BFS (Busca em Largura)

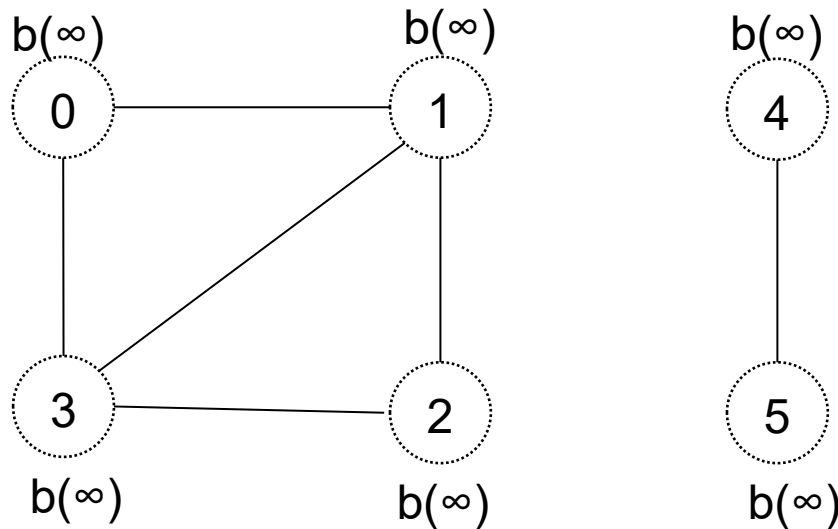
■ Exemplo detalhado

- Armazenar estados dos vértices e distância ao vértice inicial
- Estrutura de dados auxiliar: fila



BFS – passo 0

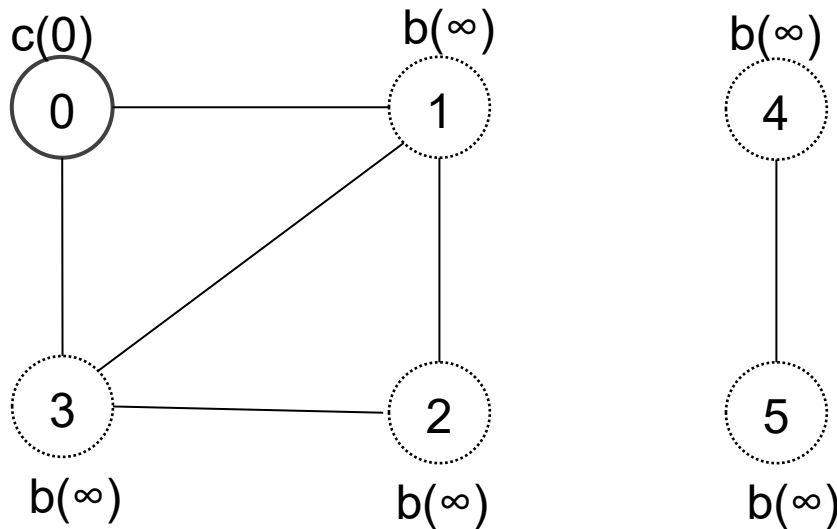
- Vértice inicial: ?; Fila = Vazia



b = branco, c = cinza, p = preto,
distância do vértice inicial entre parênteses

BFS – passo 1

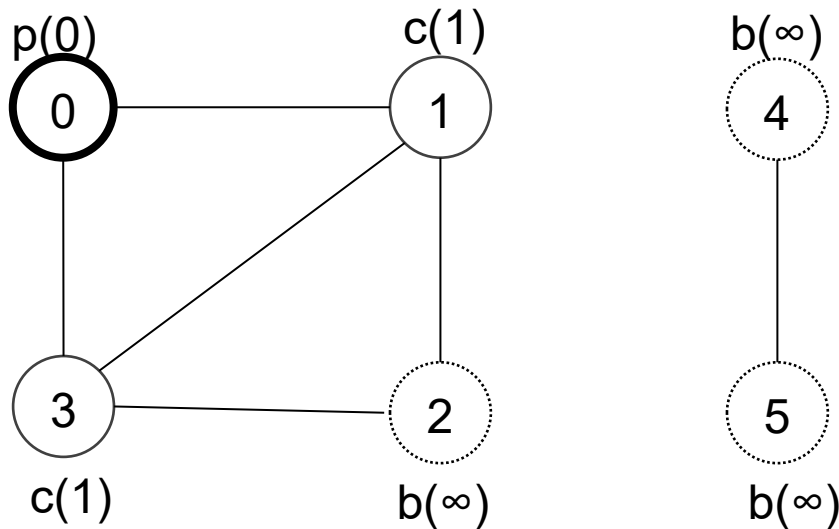
- Vértice inicial: 0; distância = 0, Fila = 0



b = branco, c = cinza, p = preto,
distância ao vértice inicial entre parênteses

BFS – passo 2

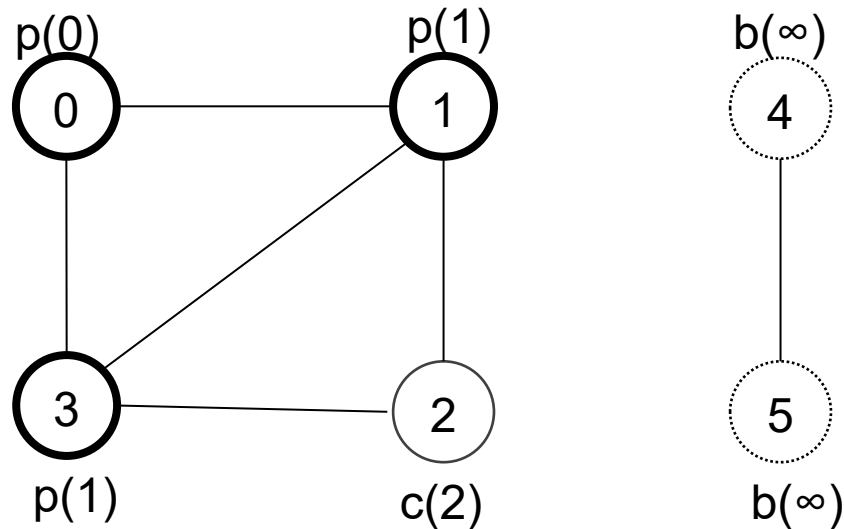
- Vértice inicial: 0; distância = 1, Fila = 1,3



Tiro da fila o vértice 0, insiro na fila seus adjacentes ainda brancos 1 e 3 (ficam cinza e atualizo distância, vértice 0 fica preto)

BFS – passo 2

- Vértice inicial: 0; distância = 2, Fila = 2

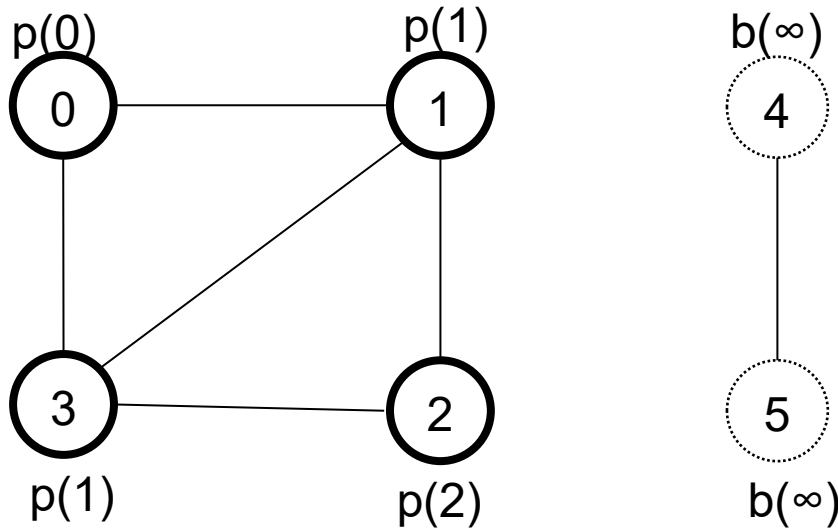


Tiro da fila o vértice 1, insiro na fila seus adjacentes ainda brancos (2: fica cinza e atualizo distância, vértice 1 fica preto),

Tiro da fila o vértice 3, insiro na fila os seus adjacentes ainda brancos (nenhum! vértice 3 fica preto)

BFS – passo 3

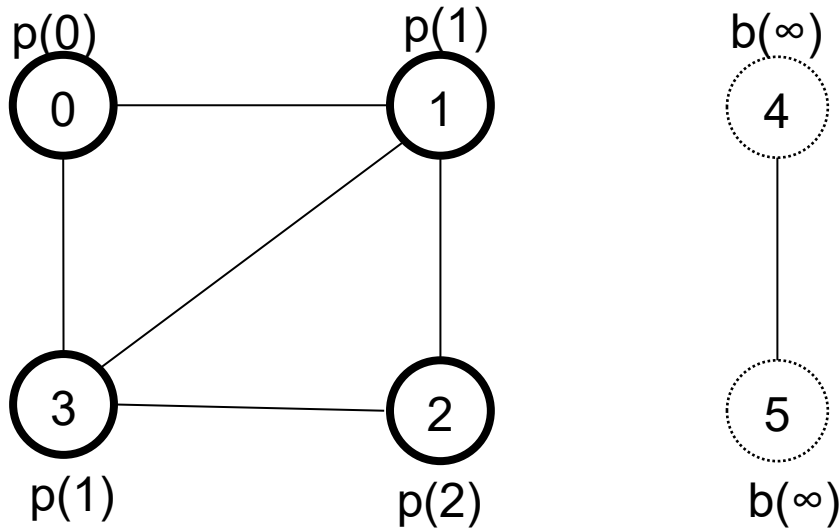
- Vértice inicial: 0; distância = 2, Fila = Vazia



Tiro da fila o vértice 2, insiro na fila seus adjacentes ainda brancos (nenhum!, vértice 2 fica preto)

BFS – passo 4

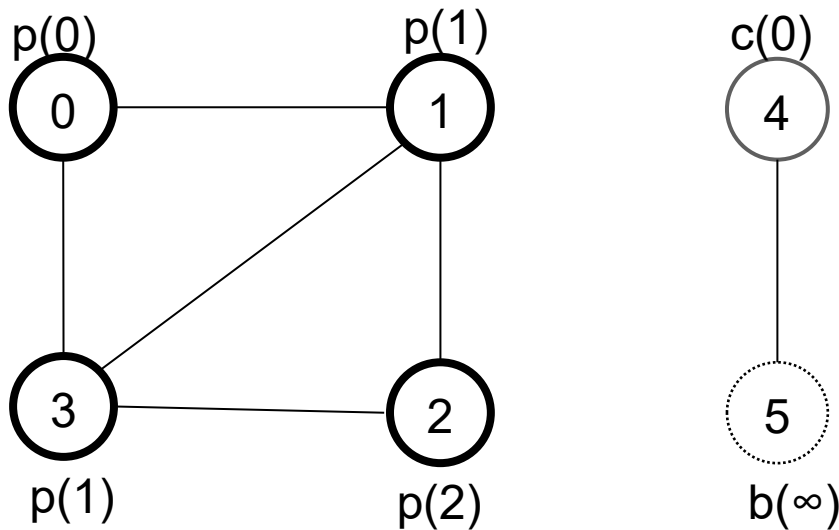
- Vértice inicial: ?; distância = 0, Fila = Vazia



Ainda tem vértices brancos, recomeço partindo de um deles...

BFS – passo 5

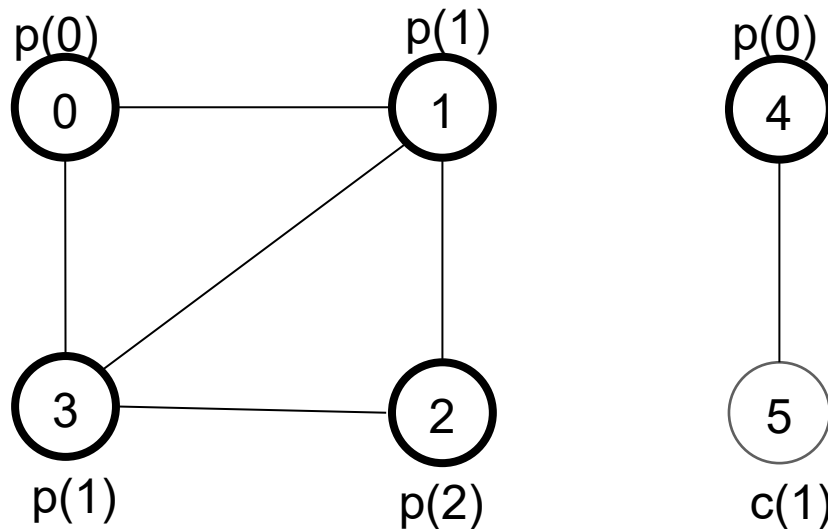
- Vértice inicial: 4; distância = 0, Fila = 4



Insiro na fila o vértice 4, fica cinza, atualizo distância

BFS – passo 6

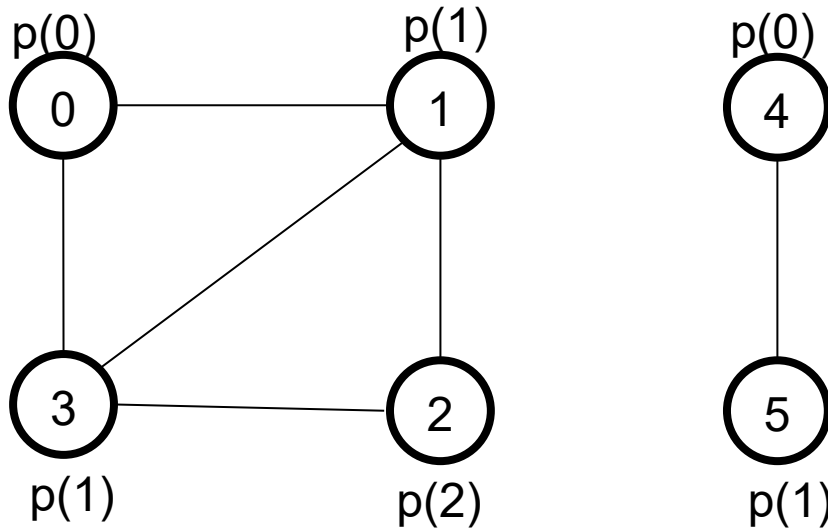
- Vértice inicial: 4; distância = 1, Fila = 5



Tiro da fila o vértice 4, insiro na fila seus adjacentes ainda brancos (5, fica cinza, atualizo distância, vértice 4 fica preto)

BFS – passo 7

- Vértice inicial: 4; distância = 1, Fila = Vazia

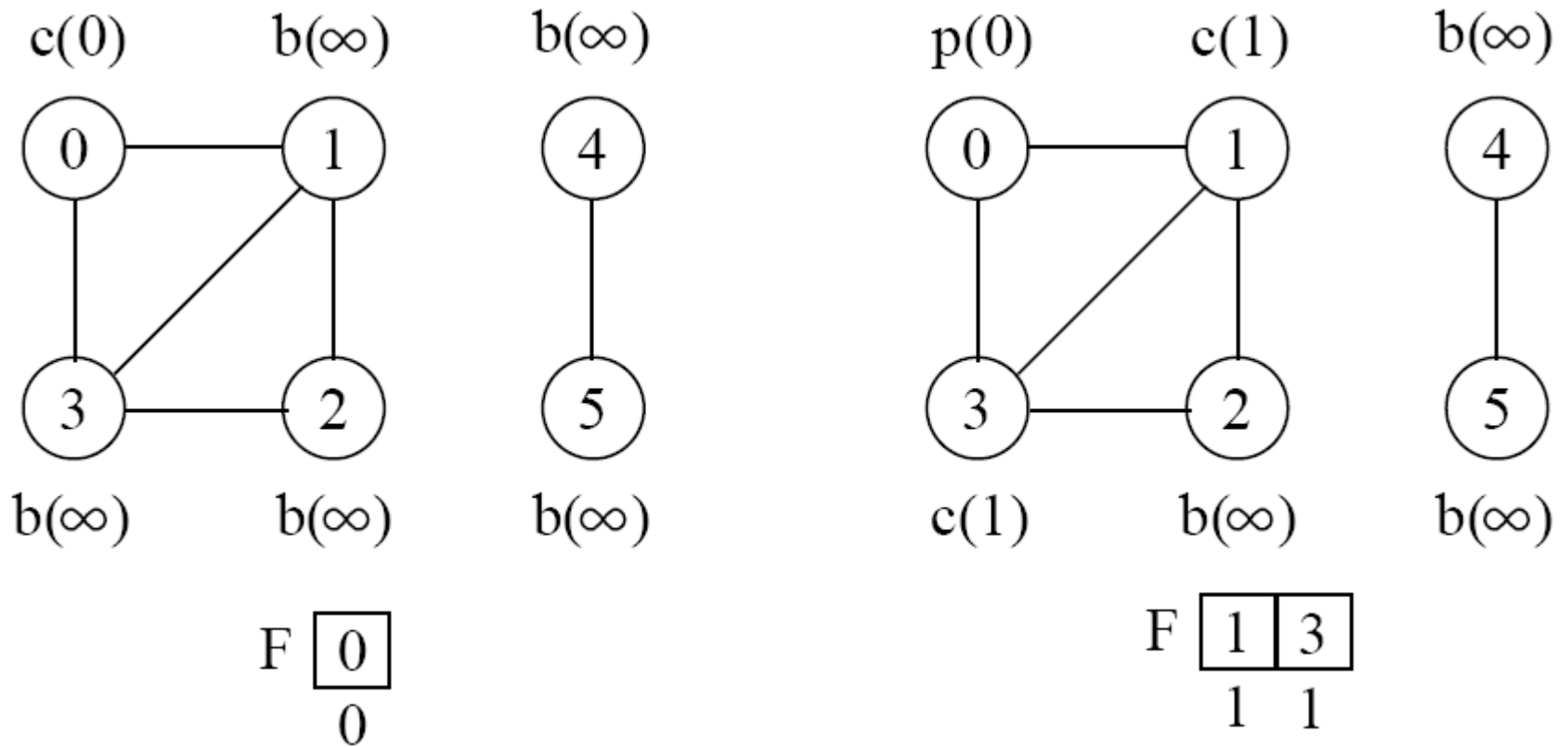


Tiro da fila o vértice 5, insiro na fila seus adjacentes ainda brancos (nenhum!)
Vértice 5 fica preto.

Todos os vértices pretos, busca encerrada

BFS (Busca em Largura)

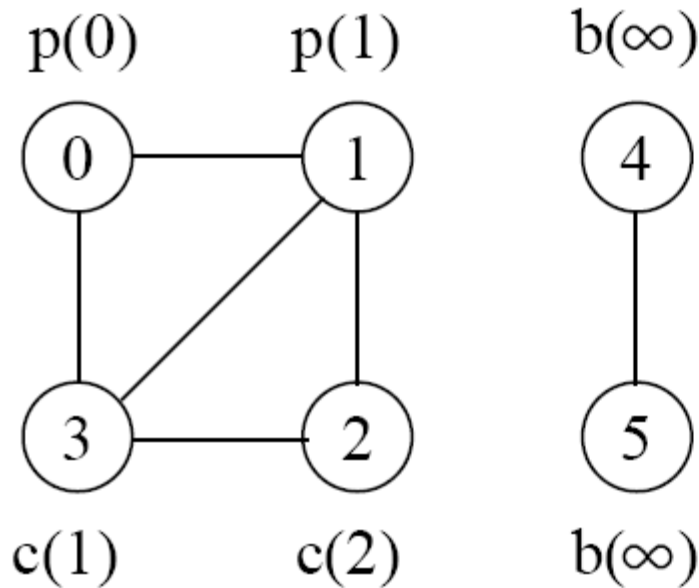
■ Exemplo detalhado



F=fila b=branco, c=cinza, p=preto, distância do vértice inicial entre parênteses

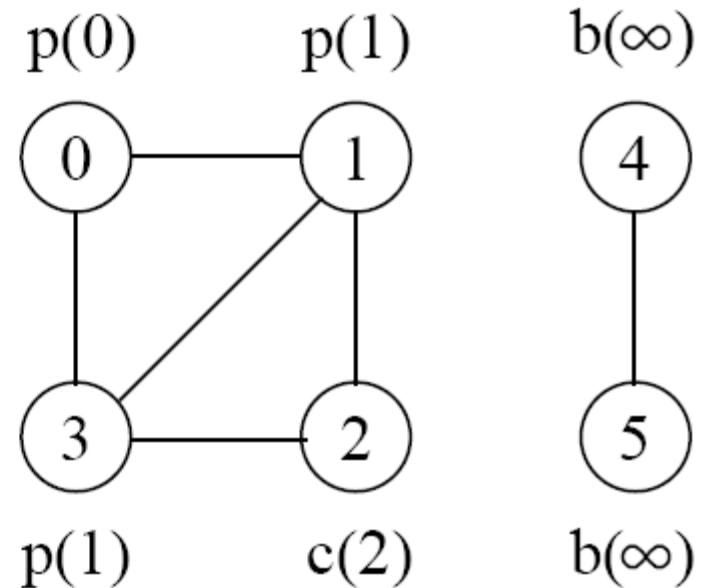
BFS (Busca em Largura)

■ Exemplo detalhado



F

3	2
1	2

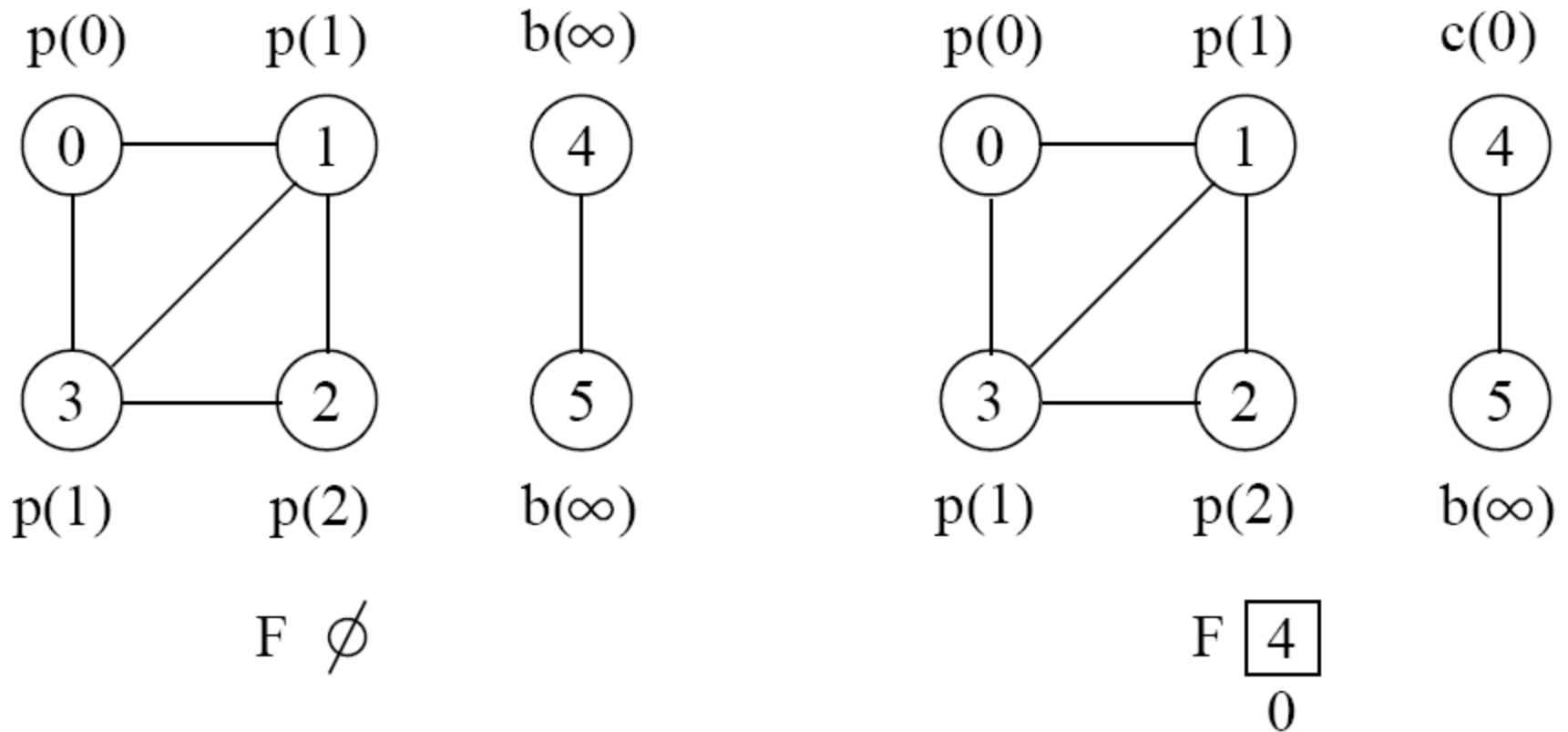


F

2
2

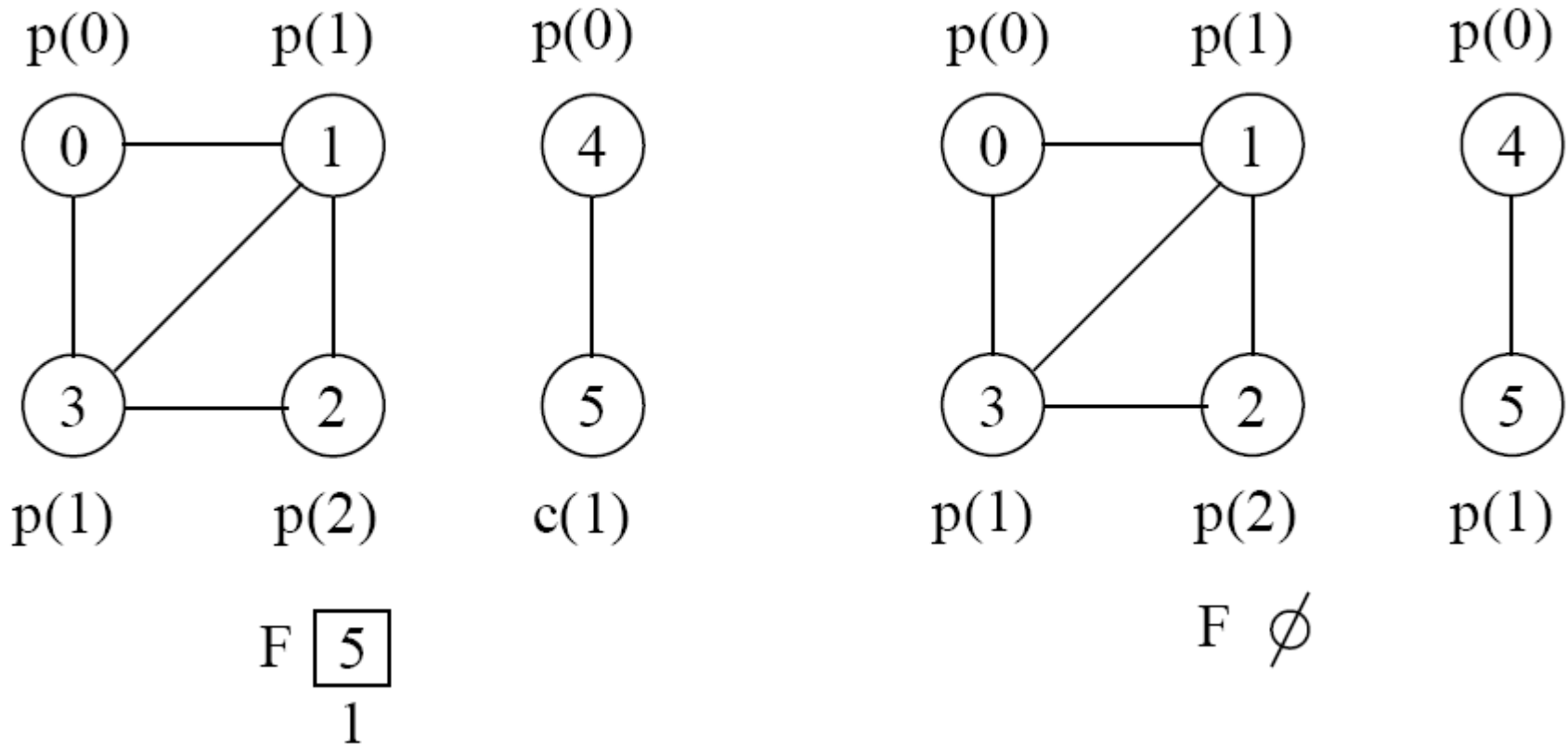
BFS (Busca em Largura)

■ Exemplo detalhado



BFS (Busca em Largura)

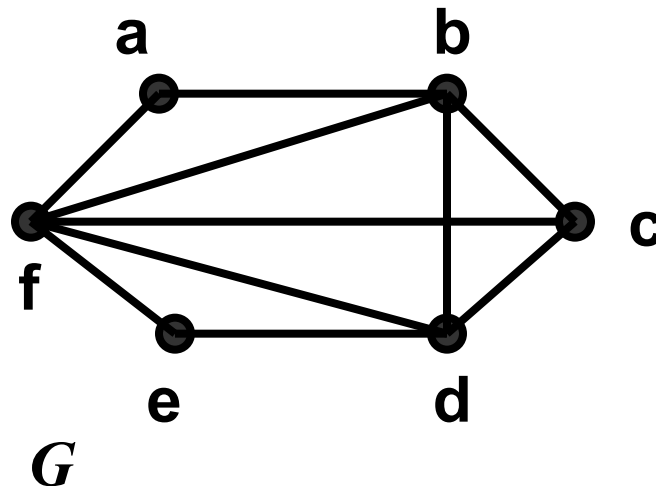
■ Exemplo detalhado



PRÁTICA

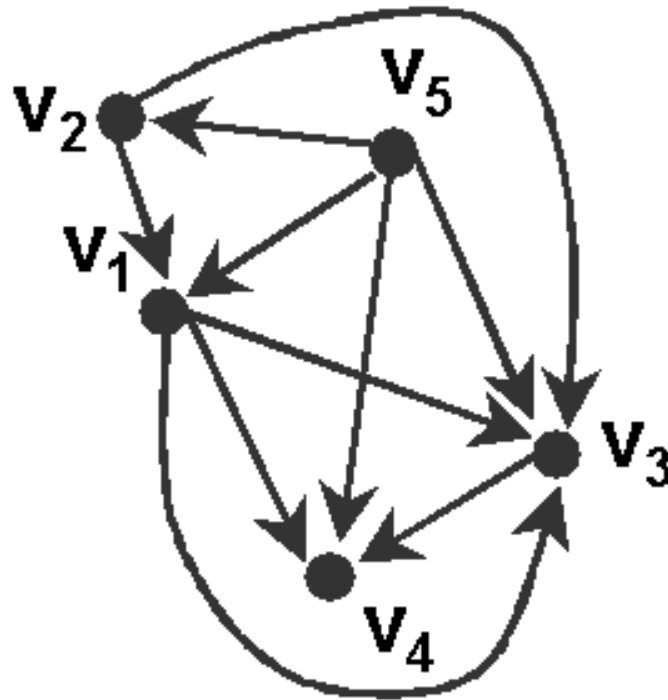
BFS (Busca em Largura)

- **Exercício:** faça a busca em largura no grafo abaixo



BFS (Busca em Largura)

- **Exercício:** faça a busca em largura no grafo abaixo



```
1 procedure BFS( $G$ ,  $start\_v$ )
2   Seja  $Q$  uma queue
3   rotule  $start\_v$  como cinza,  $dist = -1$ 
4    $Q.enqueue(start\_v)$ 
5   while  $Q$  não está vazia do
6      $v := Q.dequeue()$ ,  $dist = dist + 1$ 
7     print( $v$ ,  $dist$ )
8     for all arestas  $(v, w)$  in  $G.adjacentEdges(v)$  do
9       if  $w$  não está rotulado como cinza then
10         rotule  $w$  como cinza
11          $Q.enqueue(w)$ 
12     rotule  $v$  com preto
```

Input: Um grafo G , um vértice inicial $start_v$, todos os vertices rotulados como *branco*

Output: A sequencia de vertices rotulados como *preto*
(processados na ordem da visita BFS)

Complexidade do BFS

$O(|V| + |A|)$, ou seja, linear em relação ao tamanho da representação do grafo por listas de adjacências

- Todos os vértices são enfileirados/desenfileirados no máximo uma vez; o custo de cada uma dessas operações é $O(1)$, e elas são executadas $O(|V|)$ vezes
- A lista de adjacências de cada vértice é percorrida no máximo uma vez (quando o vértice sai da fila); o tempo total é $O(|A|)$ (soma dos comprimentos de todas as listas, igual ao número de arestas)

BFS (Busca em Largura)

- Implemente a busca em largura!

BFS (Busca em Largura)

- A busca em largura resulta no **caminho mais curto** entre o vértice inicial e um vértice qualquer do grafo!
- O procedimento `visita_bfs` constrói uma árvore de busca em largura que pode ser recuperada mantendo uma variável antecessor
 - Para cada vértice, essa variável armazena o vértice antecessor no caminho percorrido até descobri-lo (visitá-lo)

BFS (Busca em Largura)

■ Exercício

- ❑ Implemente em C uma sub-rotina que imprima os vértices do caminho mais curto entre o vértice inicial e outro vértice qualquer do grafo
- ❑ Faça a análise do algoritmo
 - ❑ Pode usar a busca em largura já realizada

BFS (Busca em Largura)

■ Questões

- ❑ *Como a busca em largura pode ajudar a determinar o número de componentes conexas de um grafo?*
- ❑ *Como alterar sua implementação para fazer isso?*