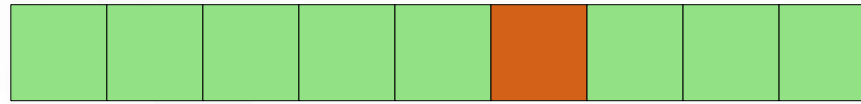




Da aula anterior

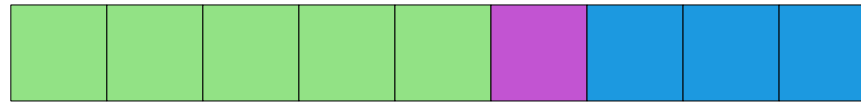
Operações em vetores

Busca →
(dado um elemento)

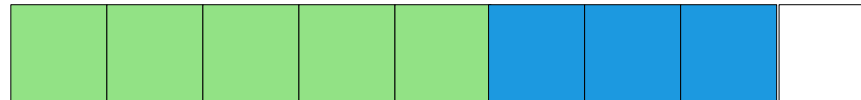


$O(n)$ ou $O(\lg n)$

Remoção →
(dado um índice)



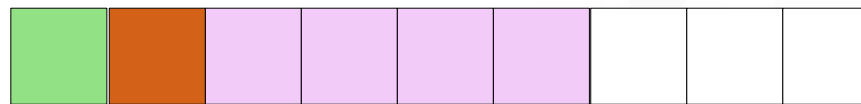
$O(n)$



Inserção →
(dado um índice
e um elemento)

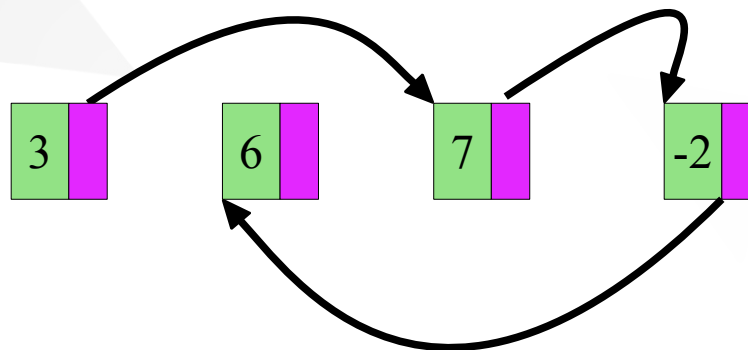


$O(n)$



Listas

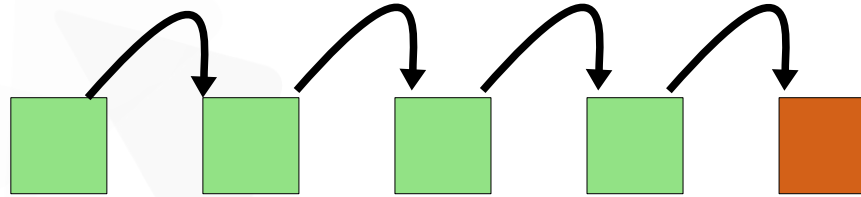
- Uma lista encadeada é uma sequência de registros que armazenam células.
 - Cada célula contém um objeto de determinado tipo.
 - Cada célula contém o endereço para a célula seguinte.



No caso da última célula, o endereço é NULL

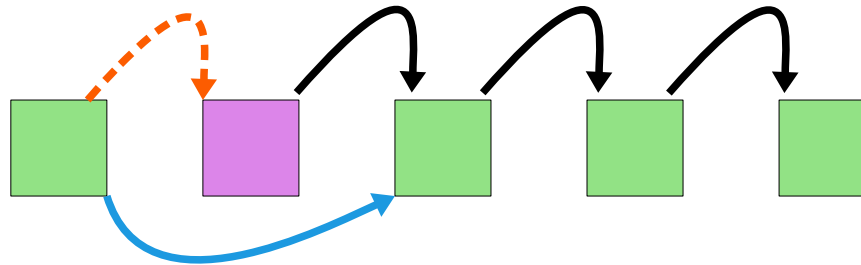
Operações considerando listas encadeadas

Busca →
(dado um elemento)



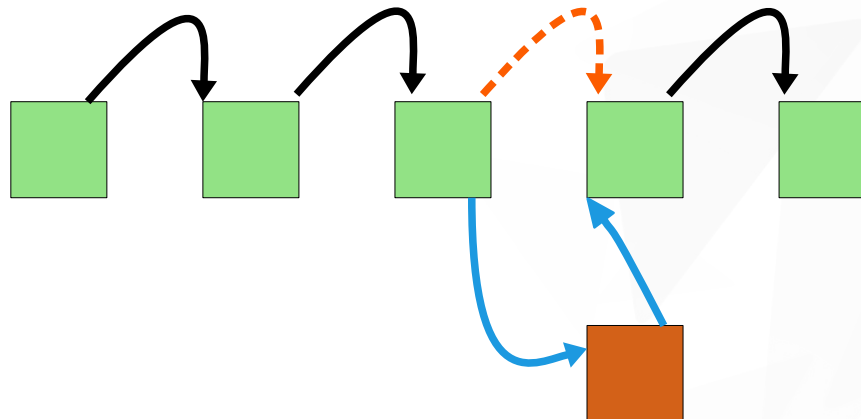
$O(n)$

Remoção →
(dado um ponteiro)



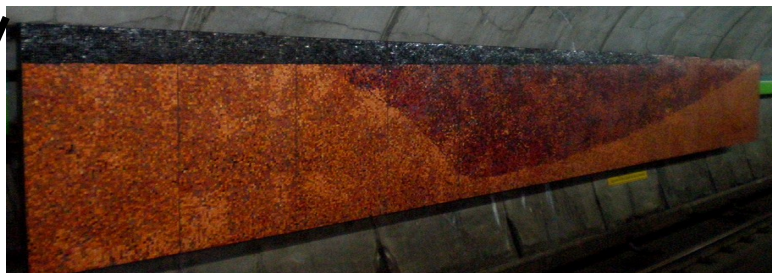
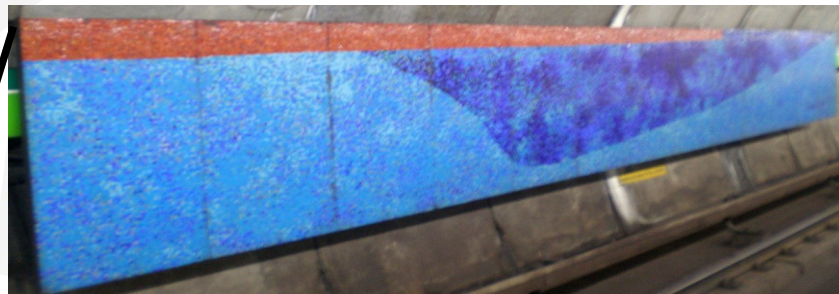
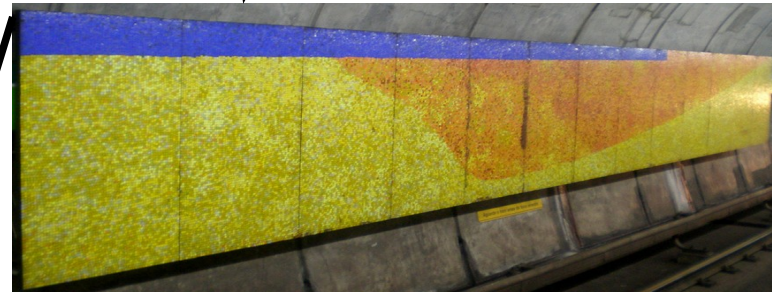
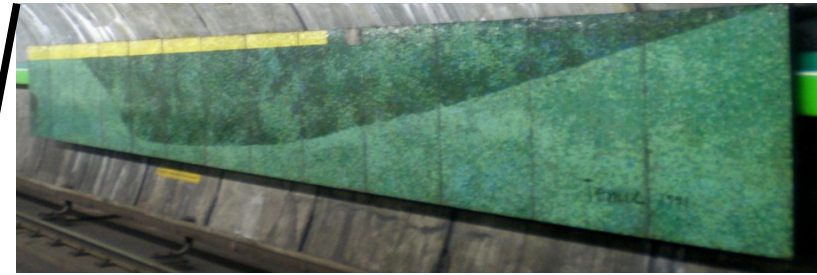
$O(1)$

Inserção →
(dado um ponteiro
e um elemento)



$O(1)$

Estação Consolação: “Quatro estações” (1991) Mosaico abstrato feito de pastilhas vitrificadas por Tomie Ohtake

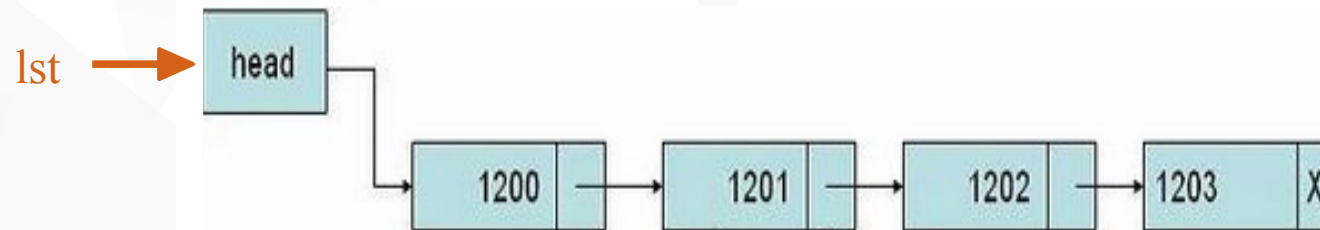


null

Listas

- Lista com cabeça:

```
celula *lst;  
lst = (celula *) malloc(sizeof(celula));  
lst->seg = NULL;
```



- Lista sem cabeça:

```
celula *lst;  
lst = NULL;
```



- Se p é um endereço para uma célula:
 - como acessar ao conteúdo dessa célula?
 - como obter o endereço da célula seguinte?



```
celula c;  
celula *p;
```

```
p = &c;
```

- Se p é um endereço para uma célula:
 - como acessar ao conteúdo dessa célula?
 - como obter o endereço da célula seguinte?



```
celula c;  
celula *p;  
  
p = &c;
```

```
(*p).conteudo  
(*p).seg
```


- Se p é um endereço para uma célula:
 - como acessar ao conteúdo dessa célula?
 - como obter o endereço da célula seguinte?



```
celula c;  
celula *p;
```

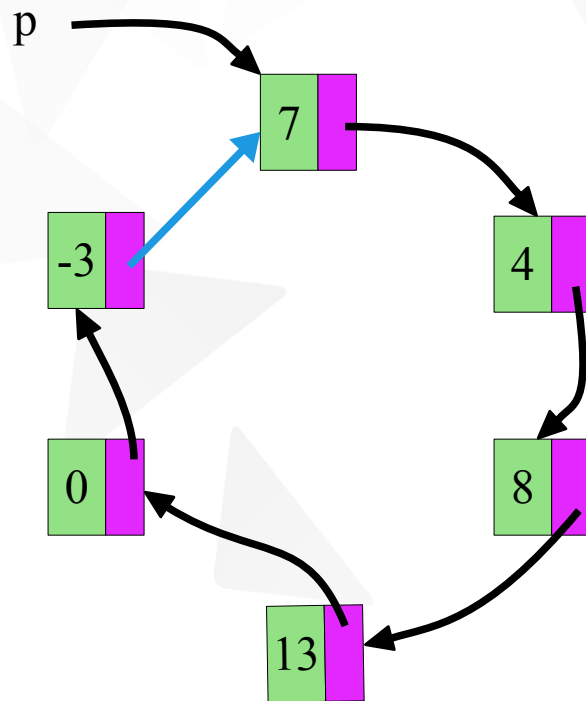
```
p = &c;
```

```
(*p) . conteudo  
(*p) . seg
```

```
p->conteudo  
p->seg
```

Outros tipos de listas encadeadas:

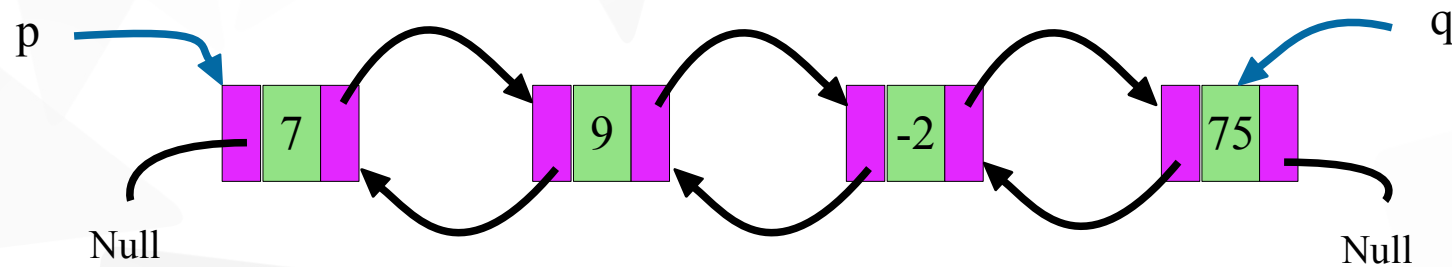
- Lista circular



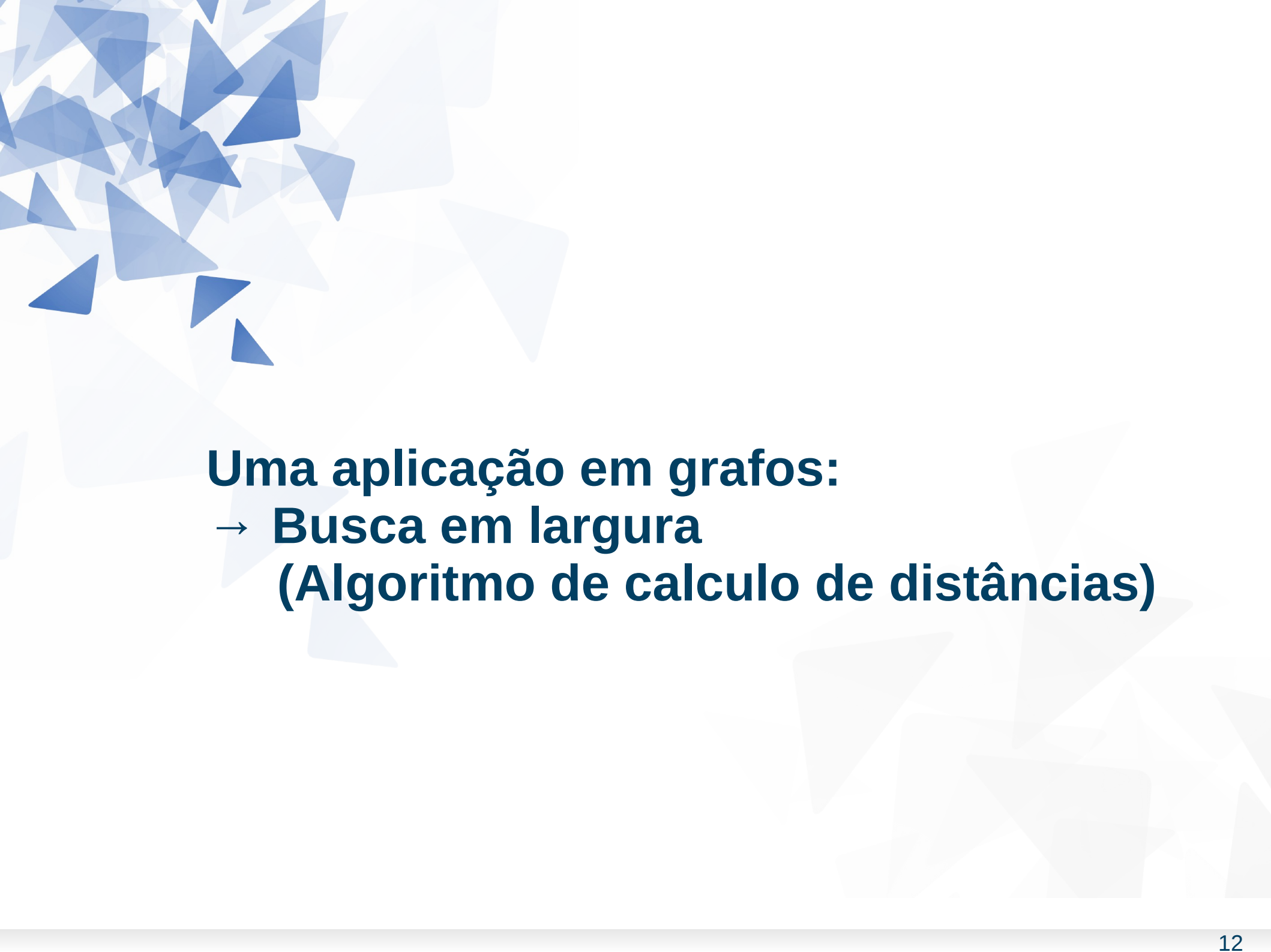
*A última célula
aponta para a primeira*

Outros tipos de listas encadeadas:

- Lista duplamente encadeada



*Cada célula contém o endereço
da célula anterior e o da
seguinte*

The background of the slide is white with a decorative pattern of blue triangles. In the top-left corner, there is a dense cluster of dark blue triangles of various sizes. Scattered across the rest of the slide are several larger, lighter blue triangles, some of which are semi-transparent, creating a layered effect.

Uma aplicação em grafos:
→ **Busca em largura**
(Algoritmo de calculo de distâncias)

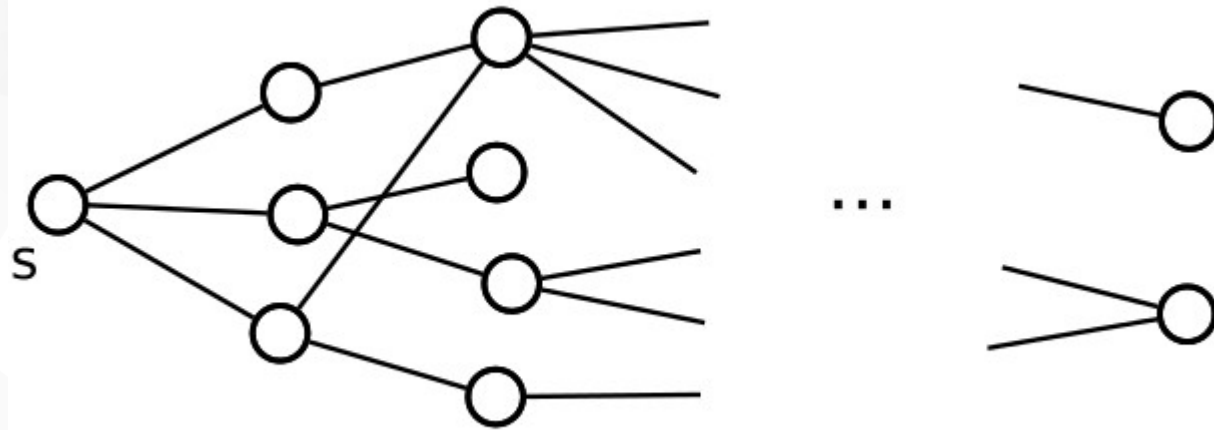
Buscas em grafos

- Em muitas aplicações de redes é necessário **percorrer rapidamente** o grafo, visitando-se **todos** os vértices.
- Para que isso seja realizado de forma **sistemática e organizada**, são utilizados **algoritmos de busca em grafos**.
- As buscas são usadas em diversas aplicações para **determinar** informações relevantes sobre a **estrutura do grafo**:
 - Web crawling
 - Redes de computadores
 - Redes sociais
 - Redes de colaboração acadêmica

Buscas em grafos

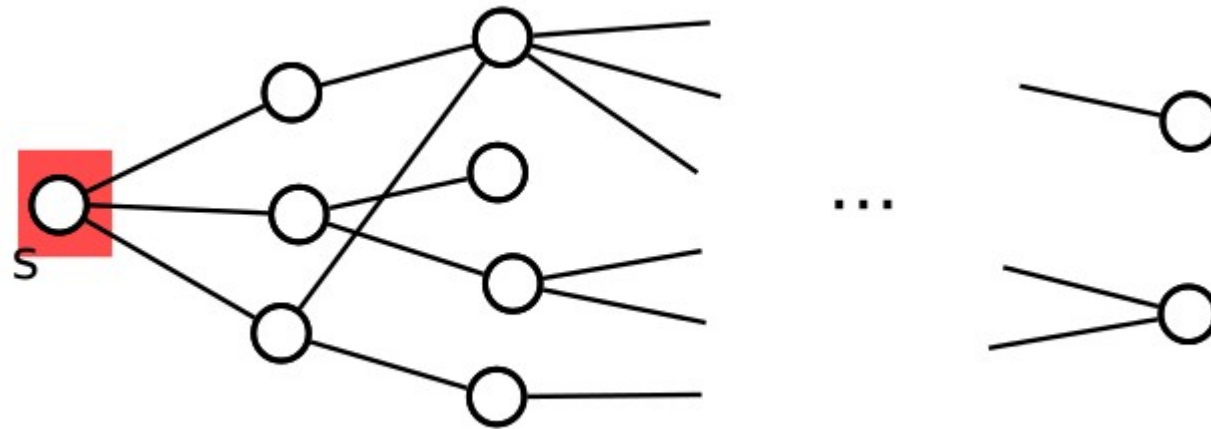
- Os algoritmos de busca em grafos permite percorrer o grafo **buscando todos os vértices que são acessíveis a partir de um determinado vértice em questão.**
- Existem diversas maneiras de realizar a busca: Cada estratégia **se caracteriza pela ordem** em que os vértices são visitados.
- São 2 os algoritmos básicos de buscas em grafos:
 - Busca em largura.
 - Busca em profundidade.

Busca em largura



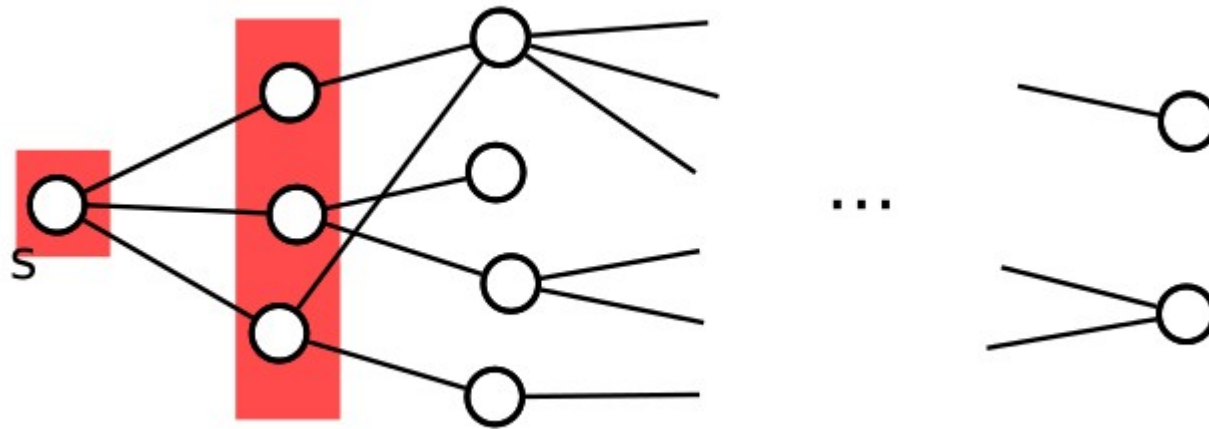
- Inicialmente **{s}**.
- Os níveis são explorados “geologicamente”.
- Fronteira = nível atual.
- Iterativamente avançar a fronteira para o nível seguinte, cuidando para não voltar ao nível inferior.

Busca em largura



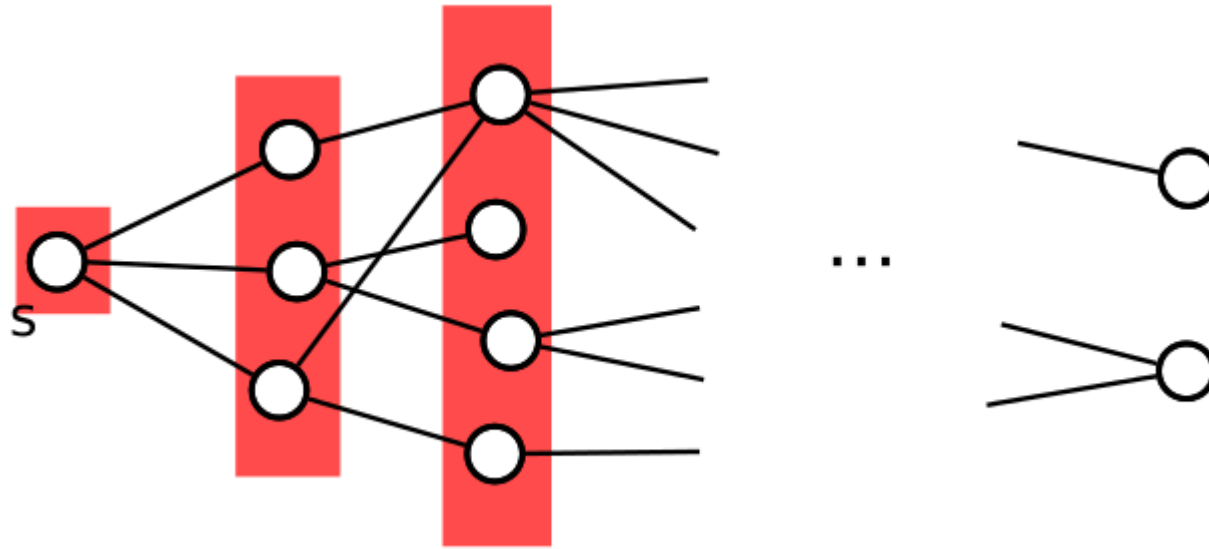
- Inicialmente $\{s\}$.
- Os níveis são explorados “geologicamente”.
- Fronteira = nível atual.
- Iterativamente avançar a fronteira para o nível seguinte, cuidando para não voltar ao nível inferior.

Busca em largura



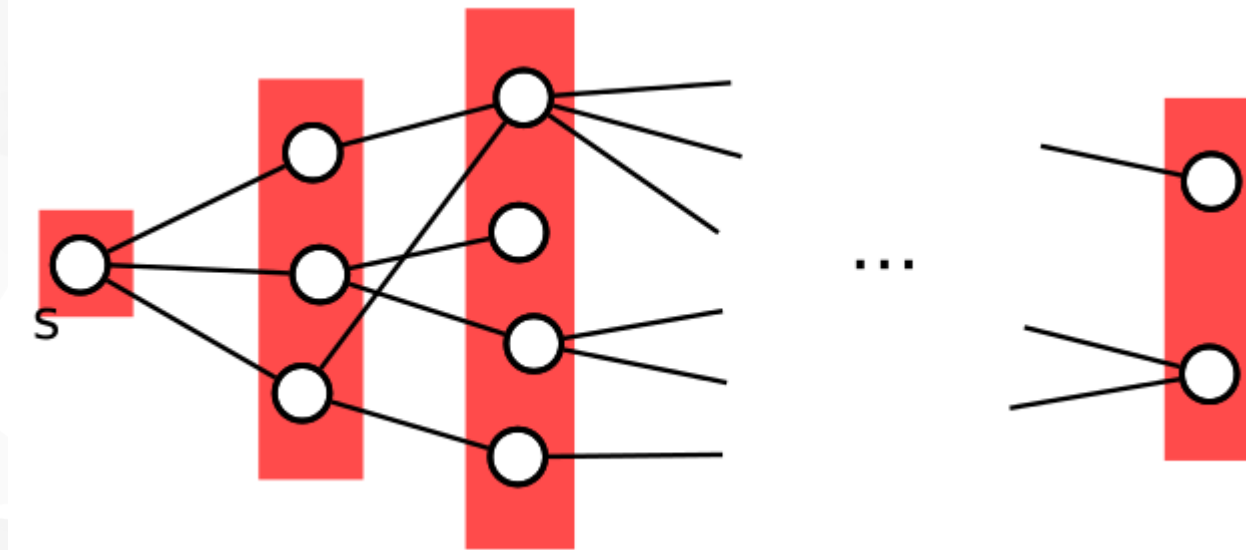
- Inicialmente **{s}**.
- Os níveis são explorados “geologicamente”.
- Fronteira = nível atual.
- Iterativamente avançar a fronteira para o nível seguinte, cuidando para não voltar ao nível inferior.

Busca em largura



- Inicialmente $\{s\}$.
- Os níveis são explorados “geologicamente”.
- Fronteira = nível atual.
- Iterativamente avançar a fronteira para o nível seguinte, cuidando para não voltar ao nível inferior.

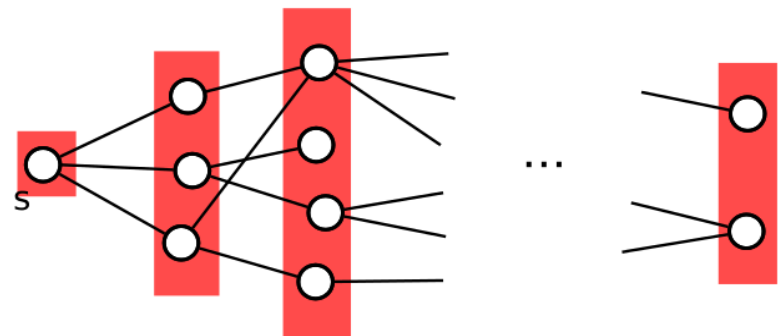
Busca em largura



- Inicialmente **{s}**.
- Os níveis são **explorados “geologicamente”**.
- Fronteira = nível atual.
- Iterativamente avançar a fronteira para o nível seguinte, cuidando para não voltar ao nível inferior.

Busca em largura

- Na busca em largura **percorre-se todos os vértices alcançáveis** a partir de um vértice **s**, em ordem de distância deste.
- Vértices a mesma distância podem ser percorridos em **qualquer ordem**.
- Se constroi uma **árvore de busca em largura** com raiz em **s**. Cada caminho de **s** a um vértice **t** corresponde a um caminho mais curto de **s** a **t**.
- O processo é implementado usando uma **FILA**.



Busca em largura

- Uma **fila** é uma estrutura de dados que permite armazenar uma sequência de valores mantendo uma determinada ordem: “primeiro a entrar, primeiro a sair”.
- *First-In-First-Out (FIFO)*



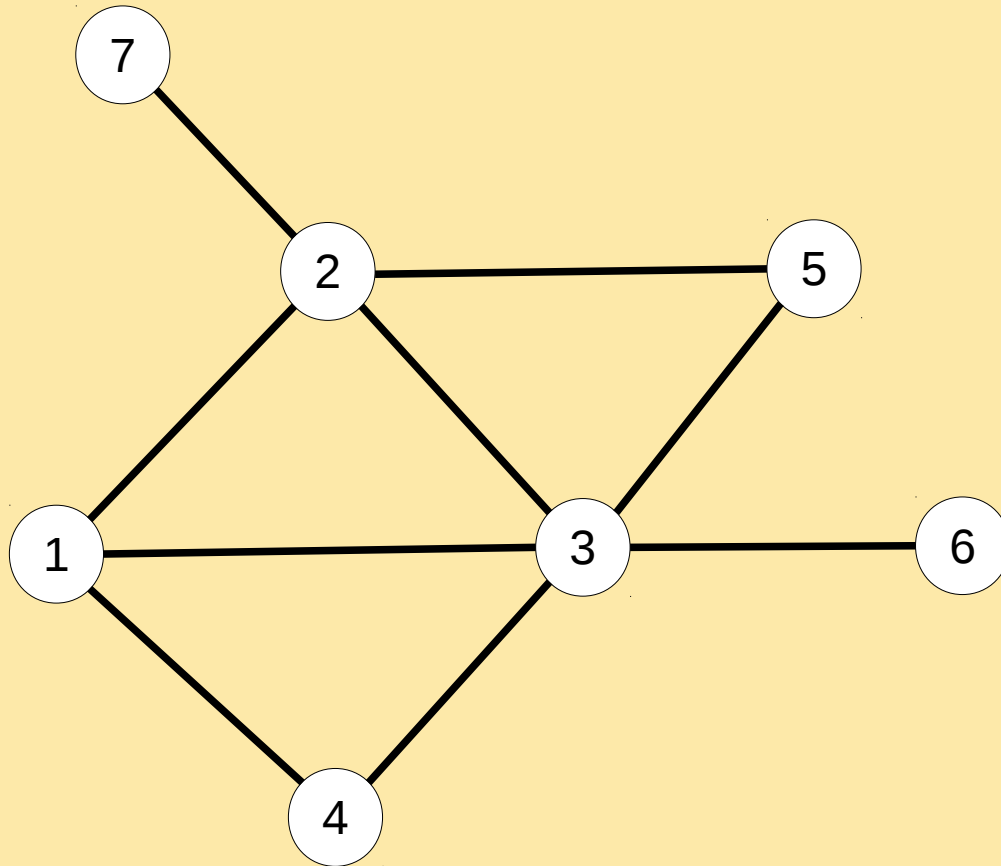
Busca em largura

O algoritmo de busca em largura **atribui cores** a cada vértice

- Cor branca = “não visitado”. Inicialmente todos os vértices são brancos.
- Cor cinza = “visitado pela primeira vez”.
- Cor preta = “teve seus vizinhos visitados”.

Busca em largura

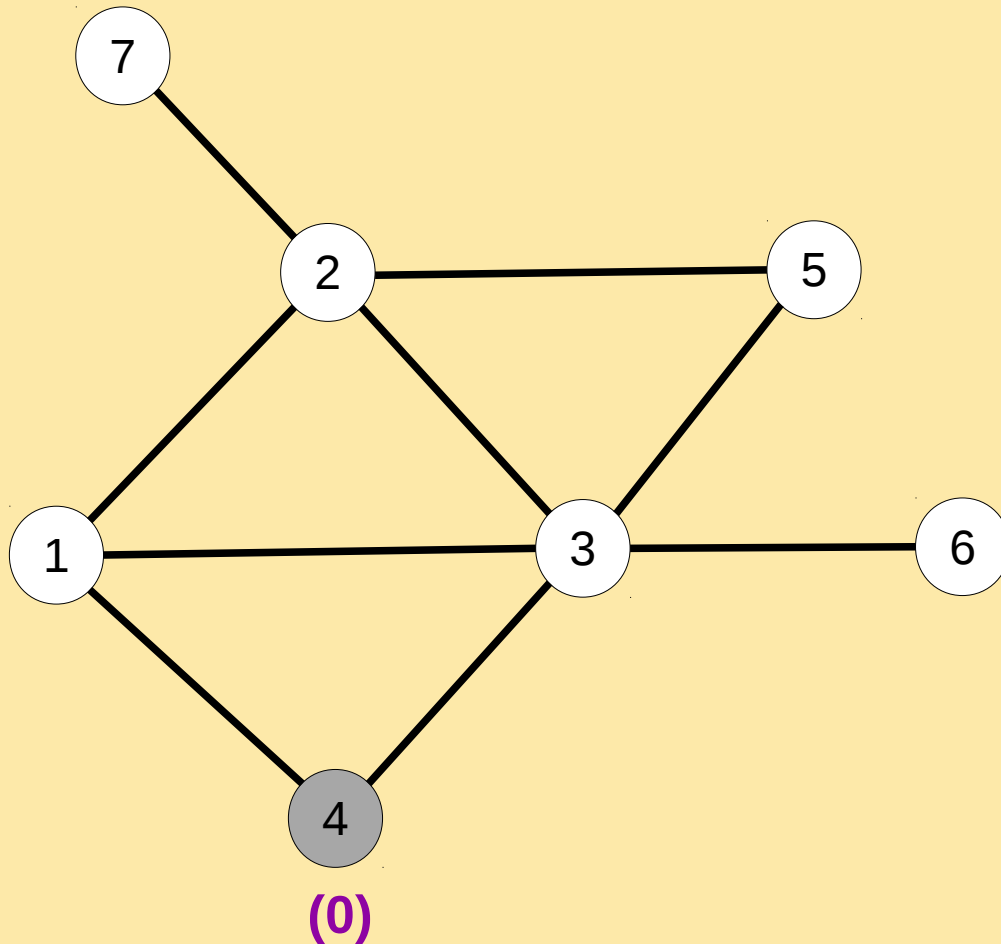
Grafo inicial



Busca em largura (origem s=4)

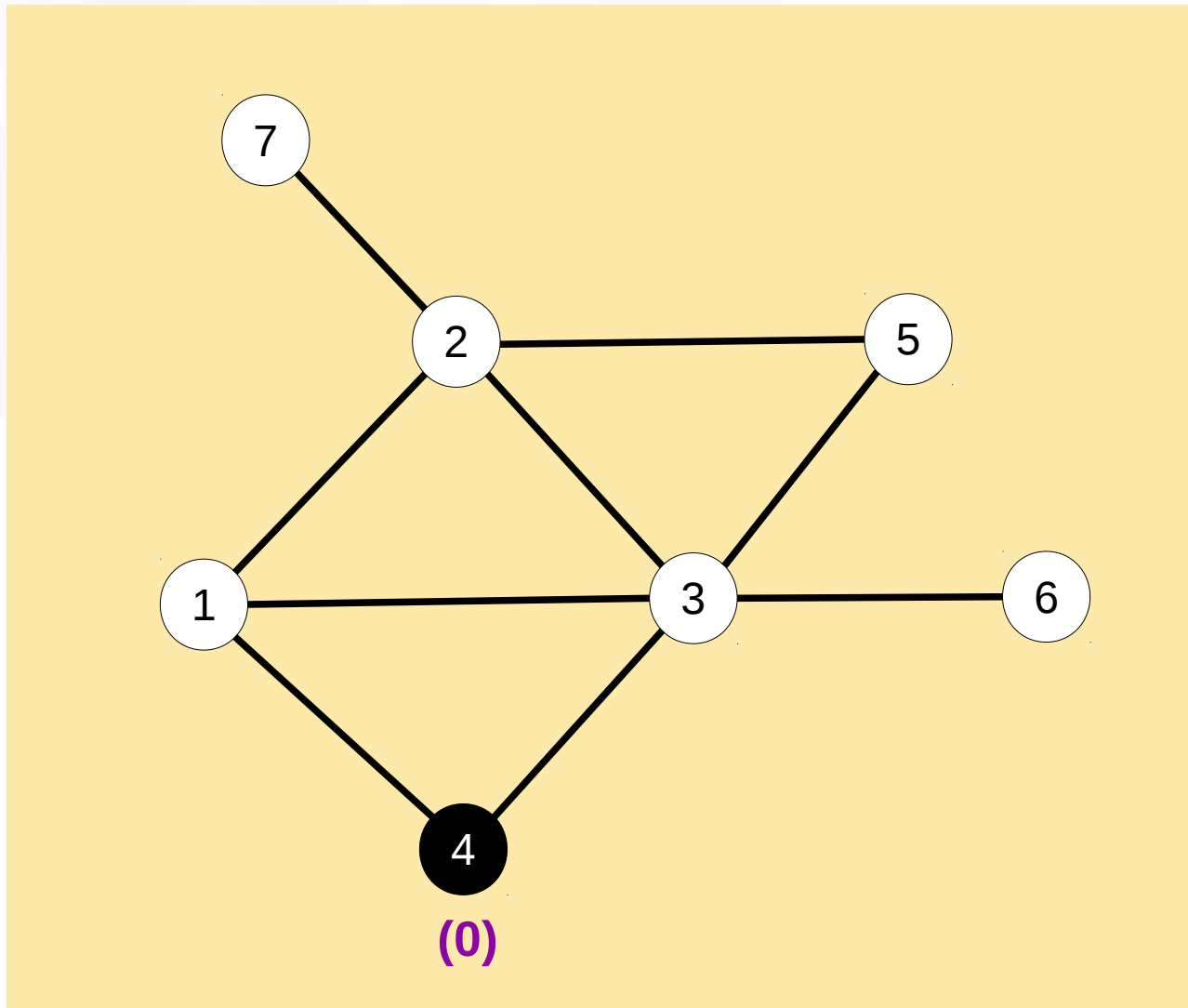
Passo 0

Fila = {4}



Busca em largura (origem s=4)

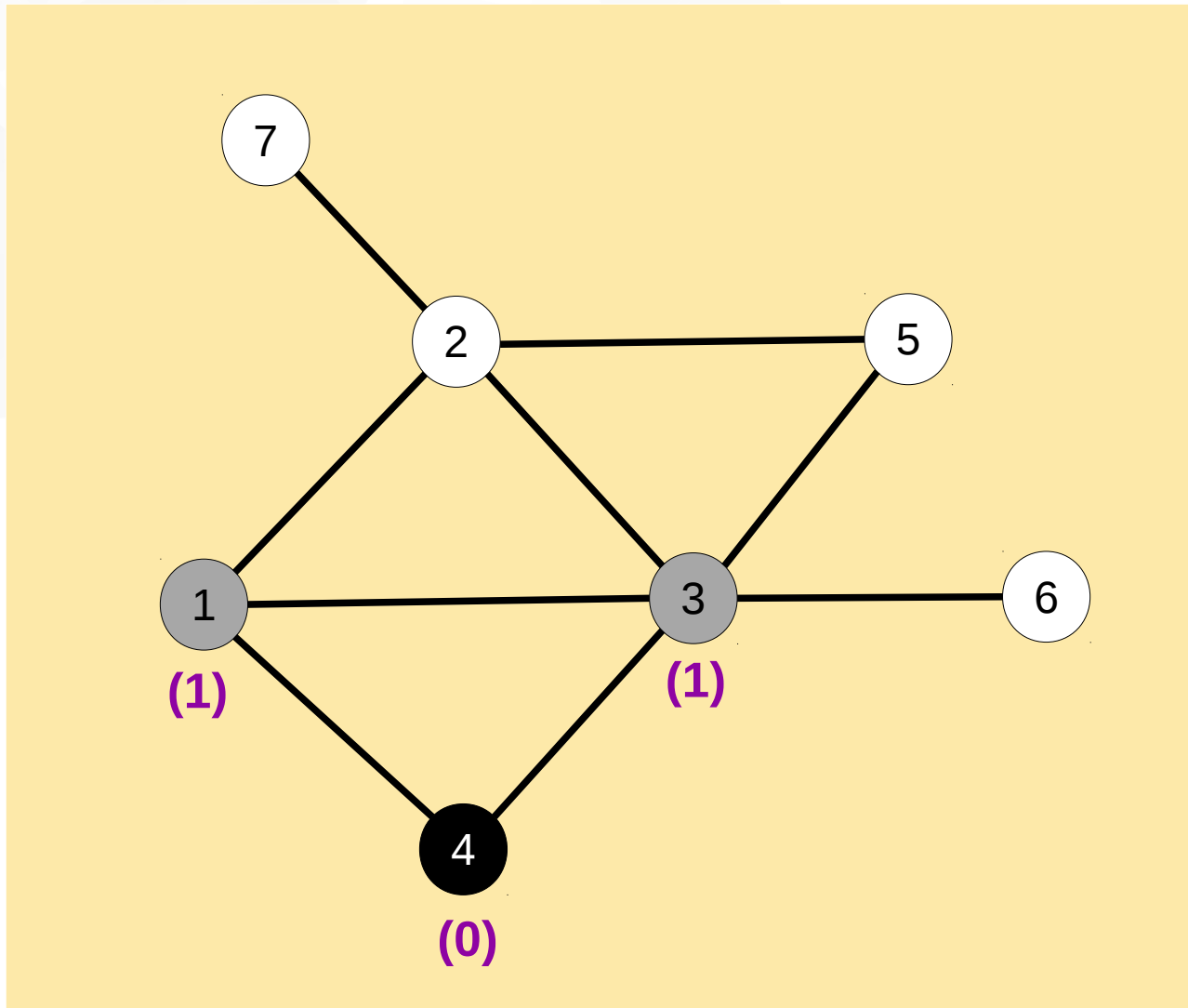
Passo 1



Fila = {}

Busca em largura (origem s=4)

Passo 2

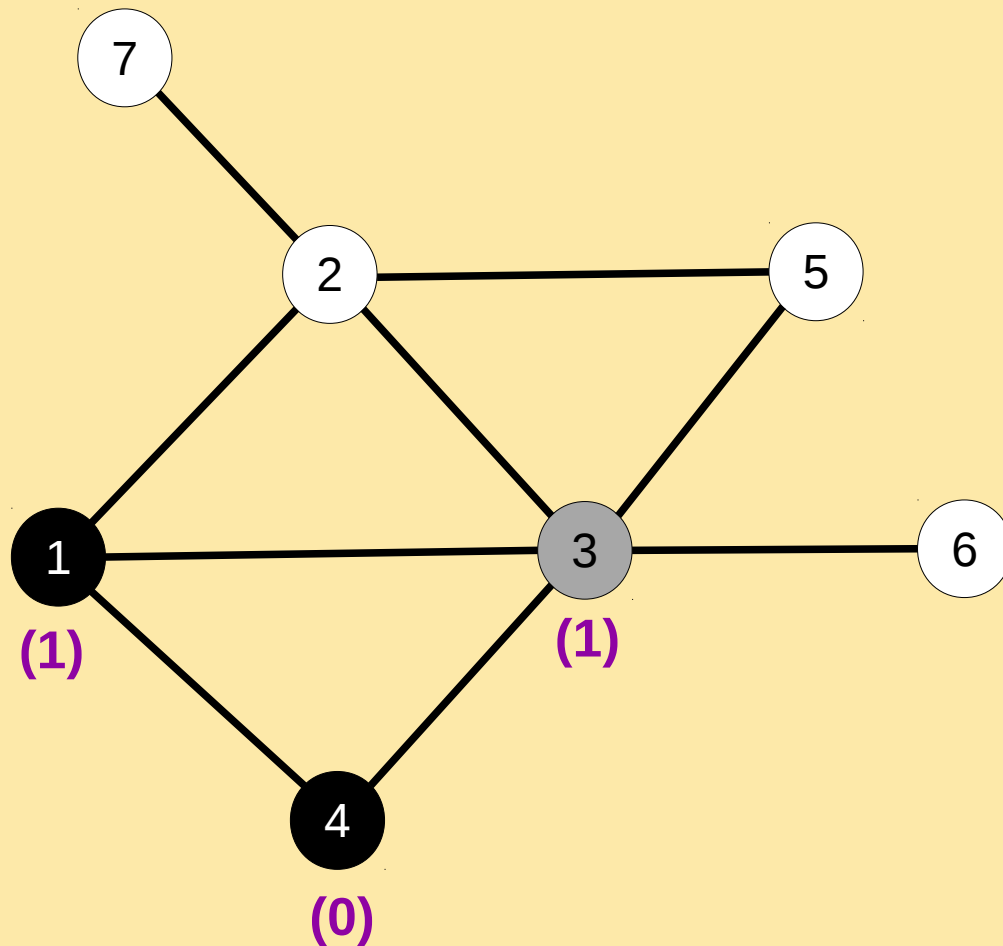


Fila = {1,3}

Busca em largura (origem s=4)

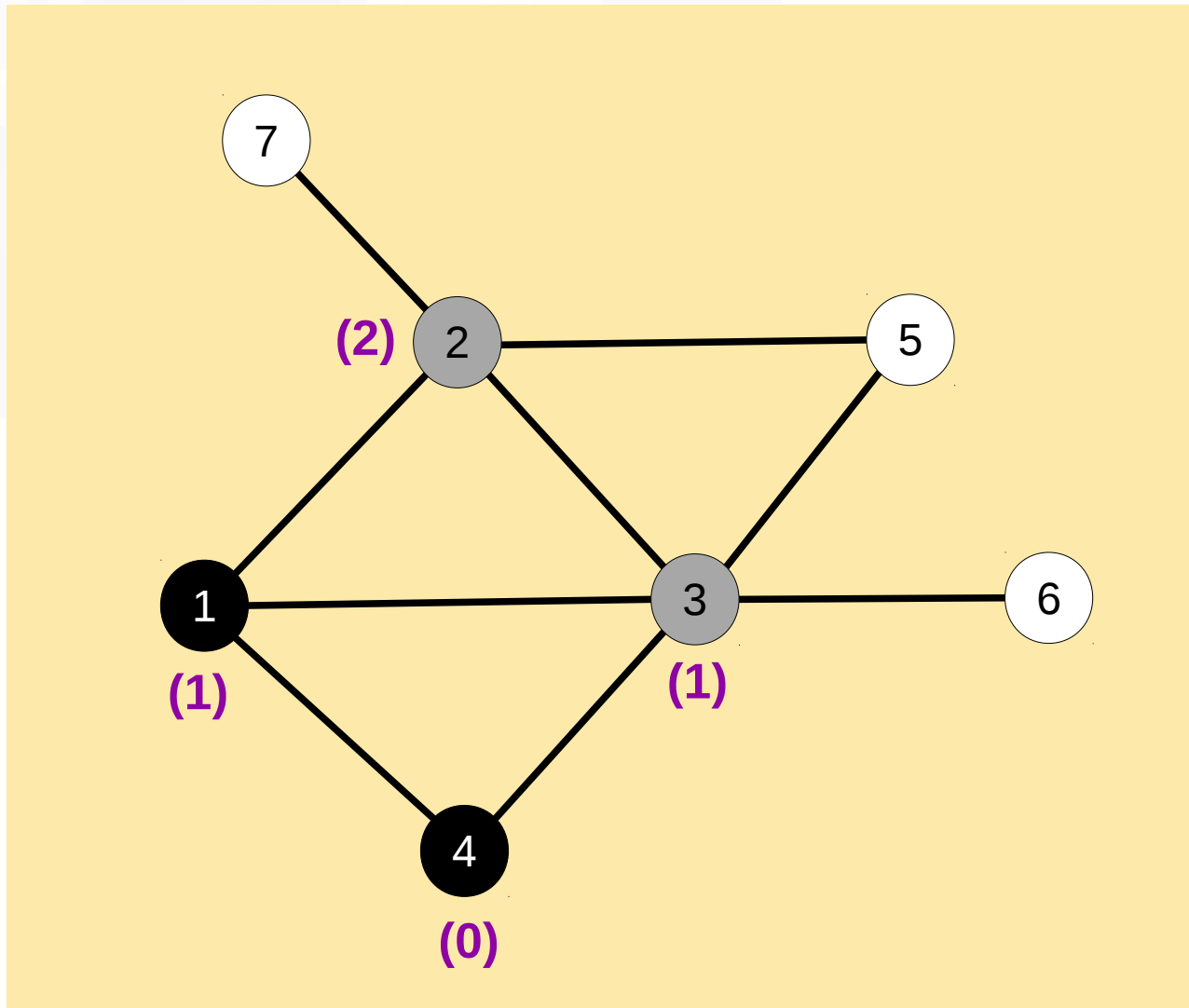
Passo 3

Fila = {3}



Busca em largura (origem s=4)

Passo 4

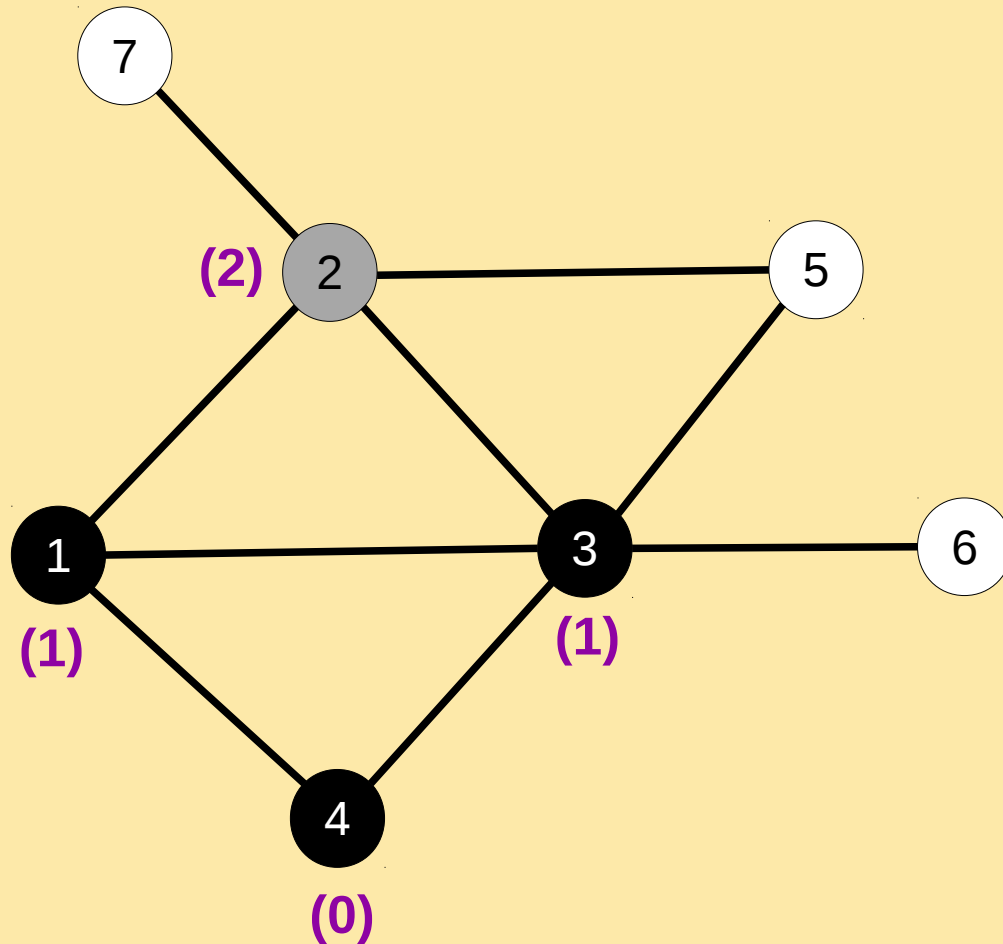


Fila = {3,2}

Busca em largura (origem s=4)

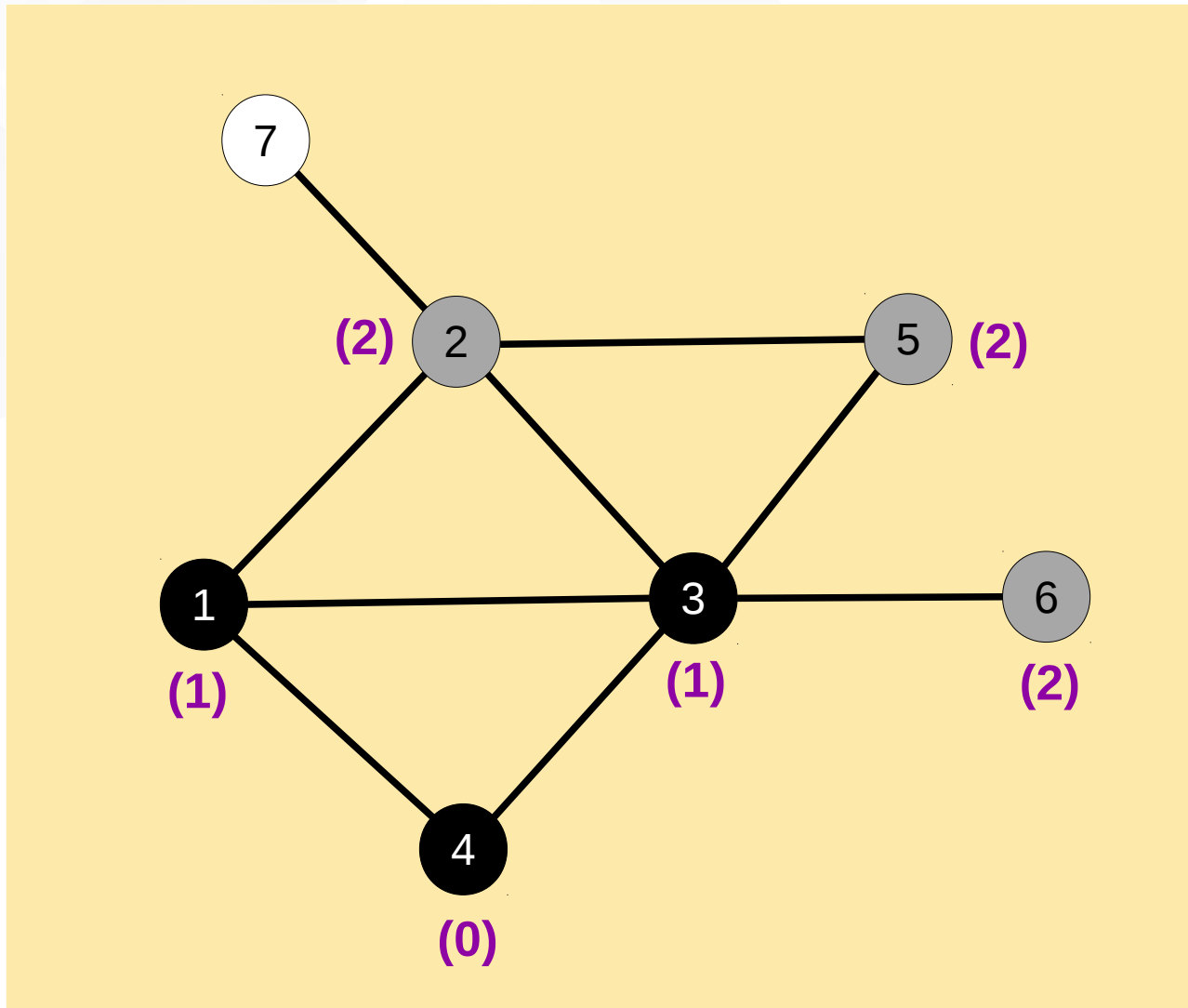
Passo 5

Fila = {2}



Busca em largura (origem s=4)

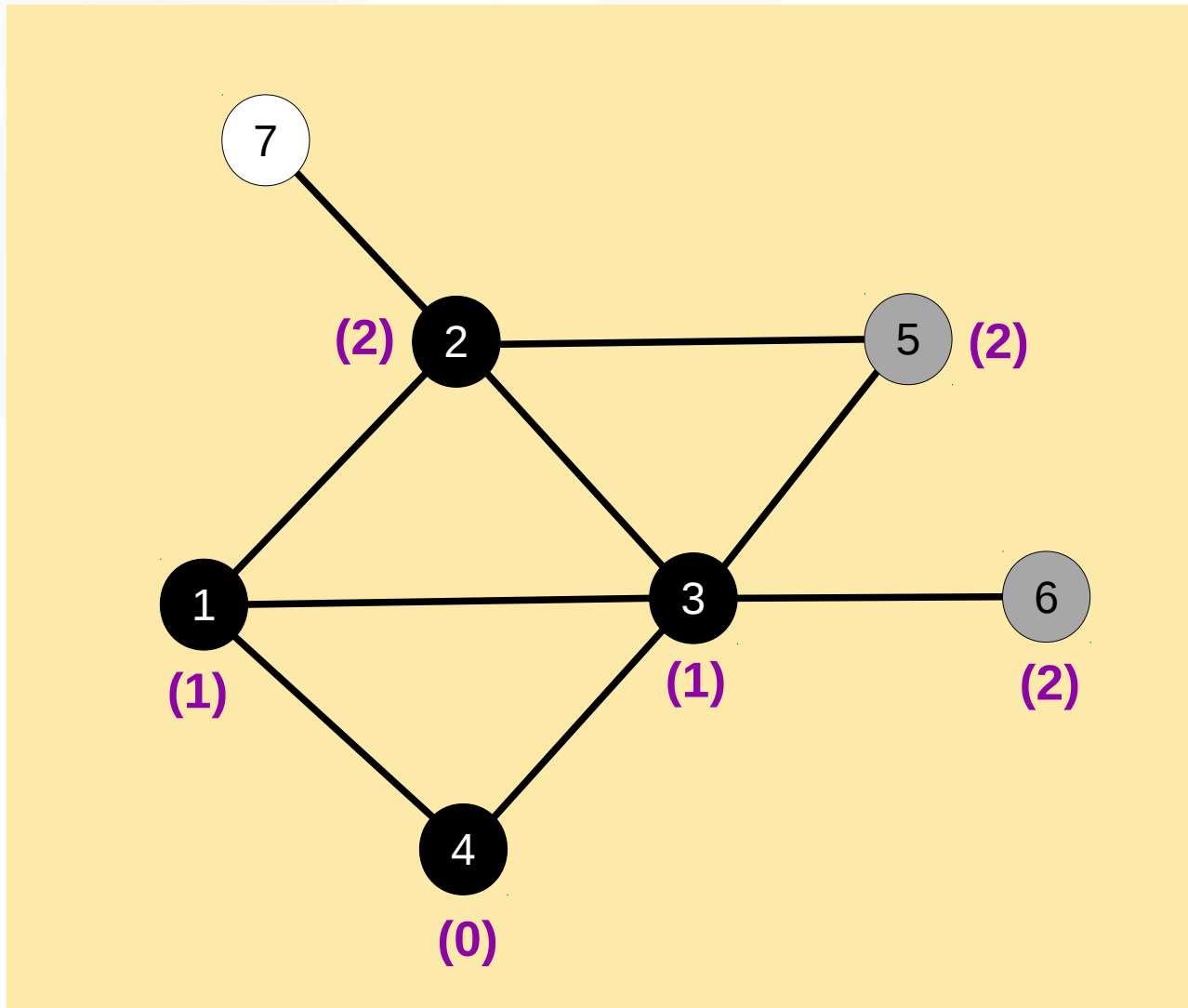
Passo 6



Fila = {2,6,5}

Busca em largura (origem s=4)

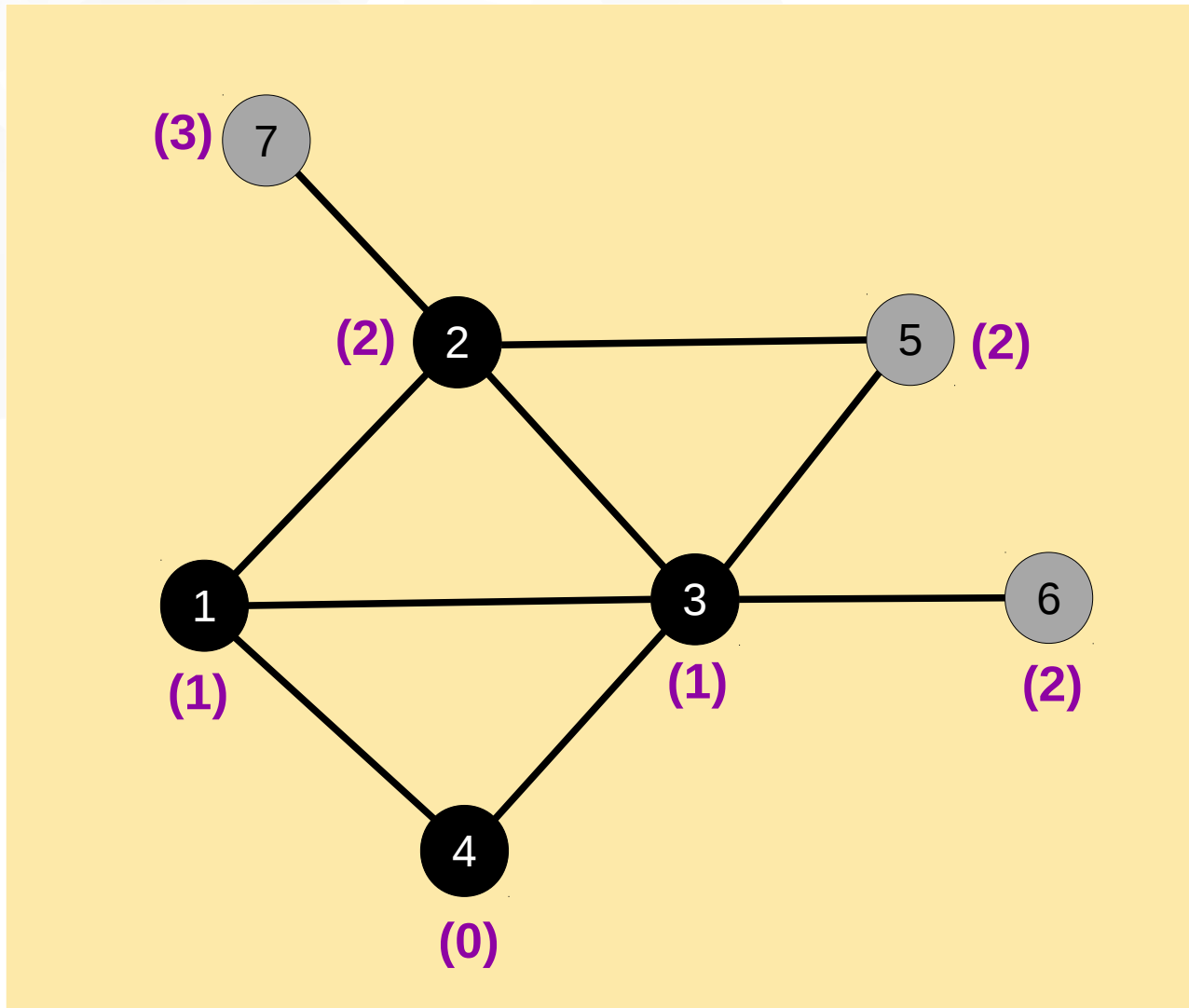
Passo 7



Fila = {6,5}

Busca em largura (origem s=4)

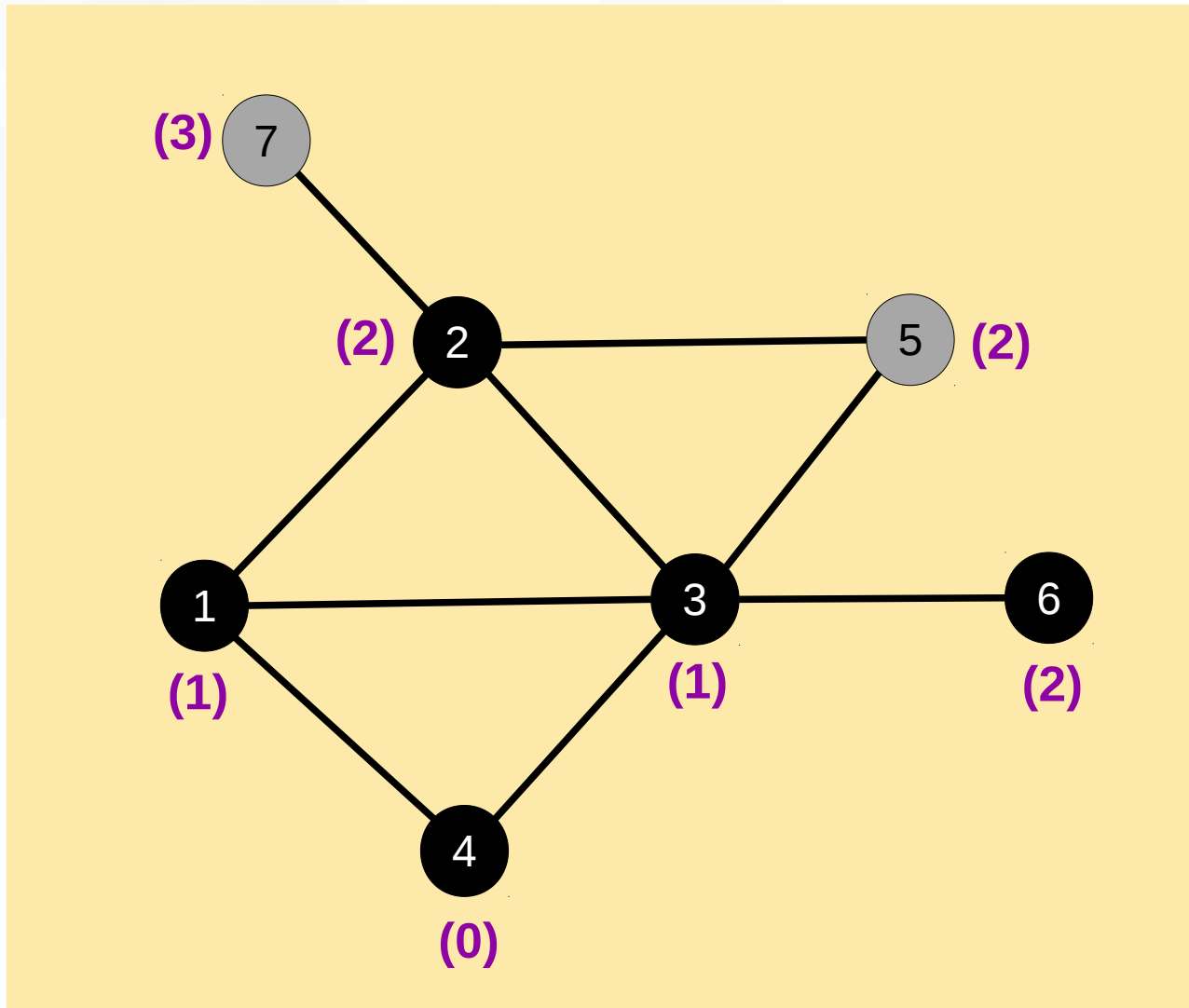
Passo 8



Fila = {6,5,7}

Busca em largura (origem s=4)

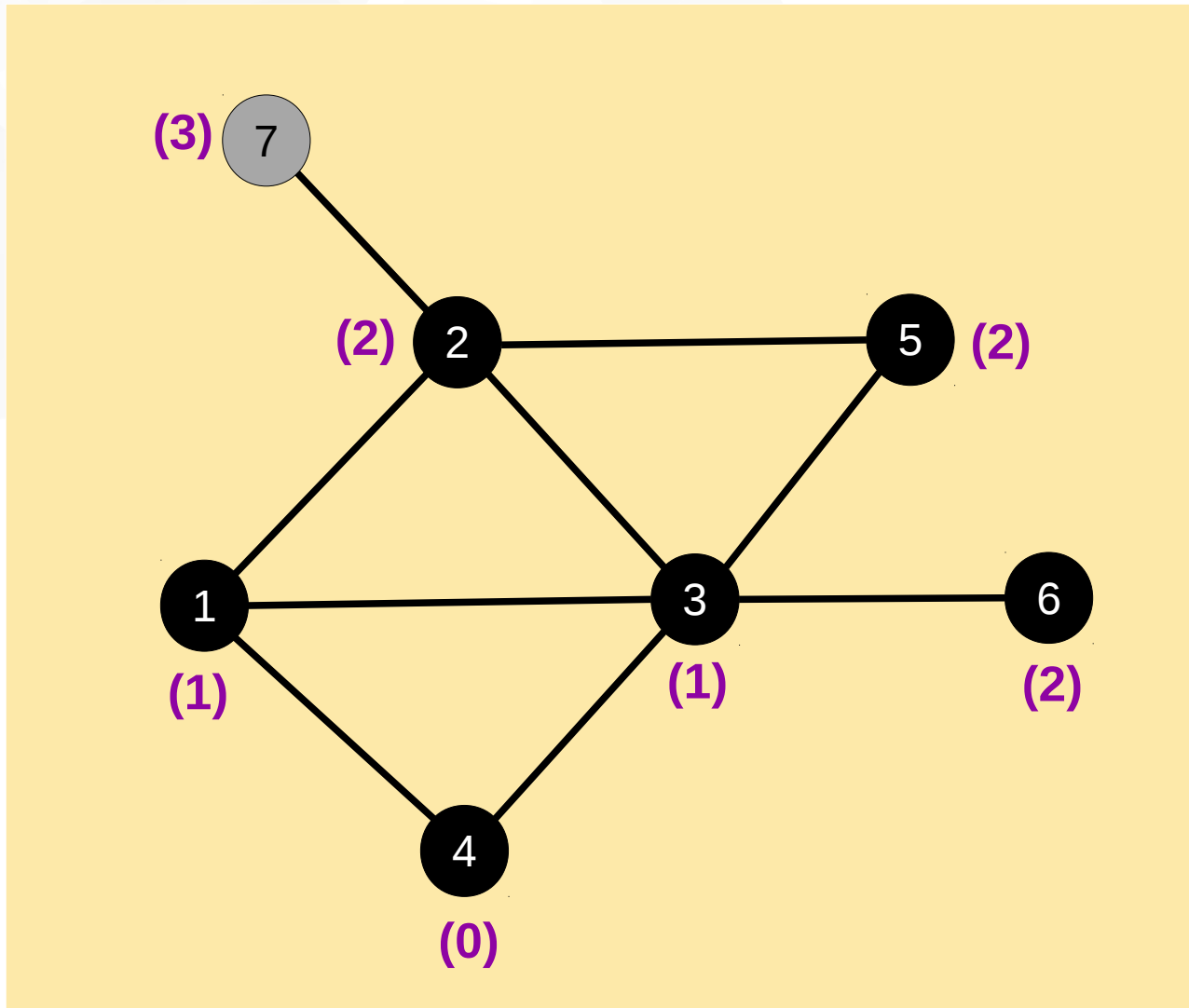
Passo 9



Fila = {5,7}

Busca em largura (origem s=4)

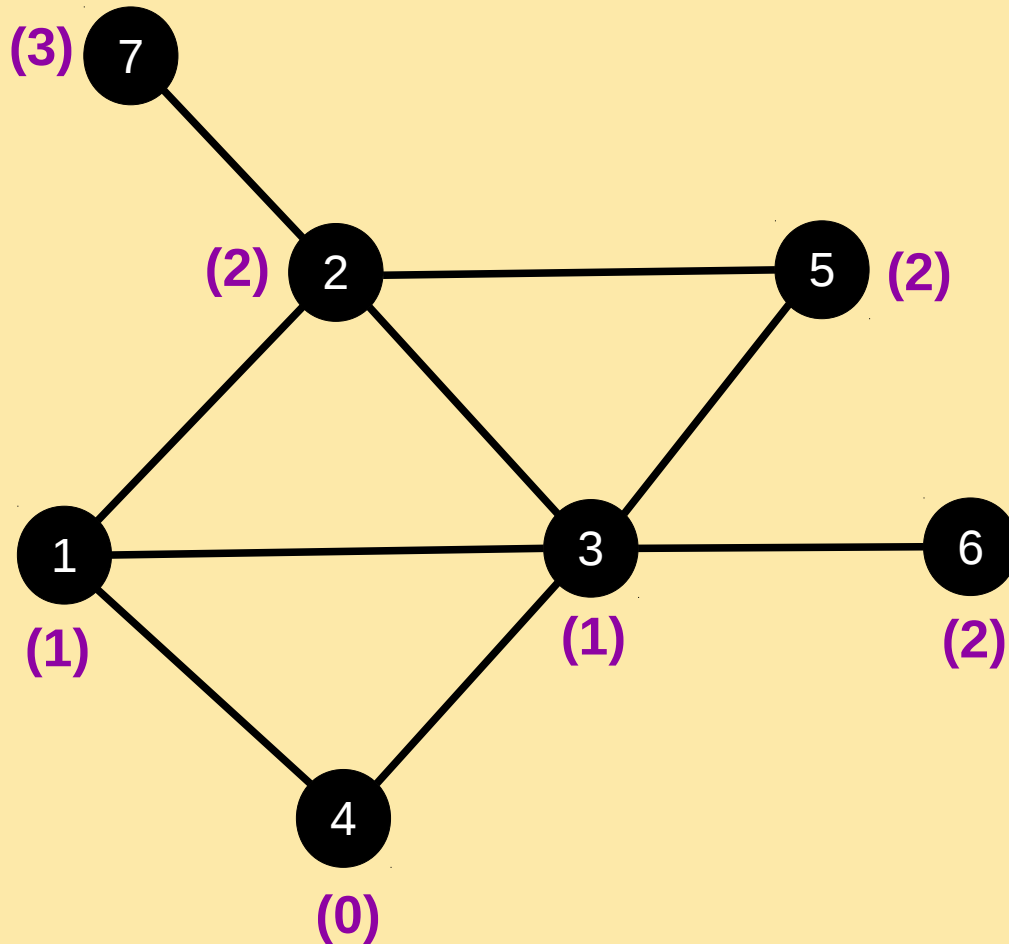
Passo 10



Fila = {7}

Busca em largura (origem s=4)

Passo 11



Fila = {}

A estrutura de dados
FILA governa a
Varredura na rede

BC1424

Algoritmos e Estruturas de Dados I

Aula 08: Filas



Prof. Jesús P. Mena-Chalco
jesus.mena@ufabc.edu.br

1Q-2015

Fila (FIFO – *First-In-First-Out*)

- Uma fila é **uma sequência de objetos dinâmica**:
 - Elementos podem ser removidos.
 - Elementos podem ser inseridos.
- Todos os elementos são do mesmo tipo.
- Regras de comportamento:
 - (1) Sempre que solicitamos a remoção de um elemento, **o elemento removido é o primeiro da sequência.**
 - (2) Sempre que solicitamos a inserção de um novo elemento, **o elemento é inserido no fim da sequência.**

Fila: Implementação em vetor

Uma fila pode ser armazenada em um segmento

$f[s..t-1]$

de um vetor

$f[0..N-1]$

Com $0 \leq s \leq t \leq N$

- O primeiro elemento está na posição de s .
- O último elemento está na posição $t-1$.
- A fila está vazia se $s==t$.
- A fila está cheia se $t==N$.

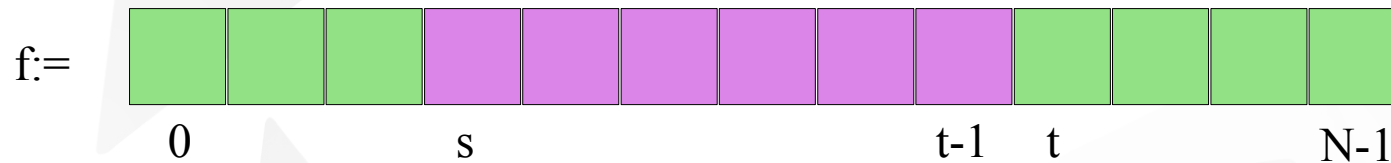
Fila: Implementação em vetor

Uma fila pode ser armazenada em um segmento

$f[s..t-1]$

de um vetor

$f[0..N-1]$



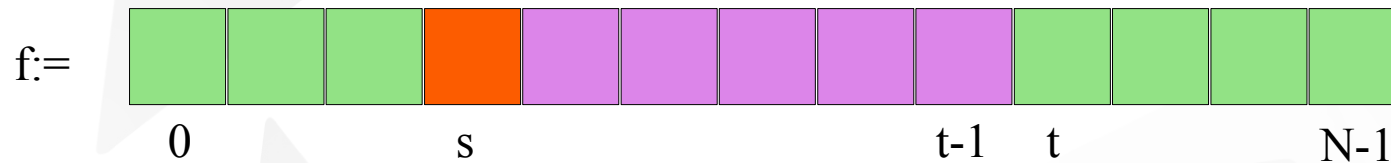
Fila: Implementação em vetor

Uma fila pode ser armazenada em um segmento

$f[s..t-1]$

de um vetor

$f[0..N-1]$



Para remover um elemento da fila f ?

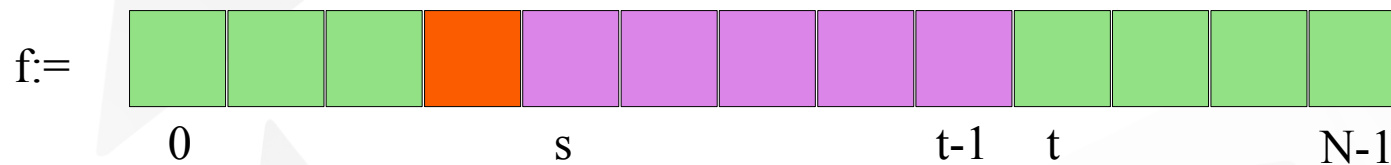
Fila: Implementação em vetor

Uma fila pode ser armazenada em um segmento

$f[s..t-1]$

de um vetor

$f[0..N-1]$



Para remover um elemento:

$x = f[s];$

$s = s+1;$

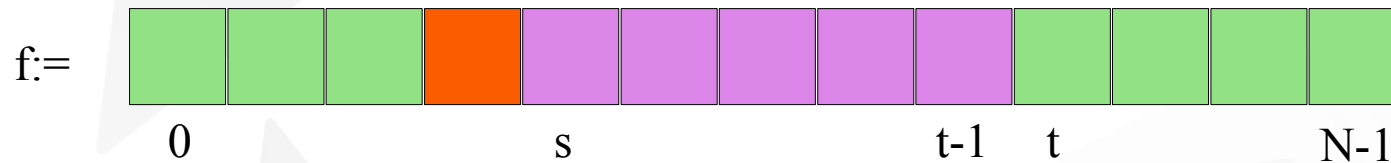
Fila: Implementação em vetor

Uma fila pode ser armazenada em um segmento

$f[s..t-1]$

de um vetor

$f[0..N-1]$



Para remover um elemento:

$x = f[s];$

$s = s+1;$

$x = f[s++];$

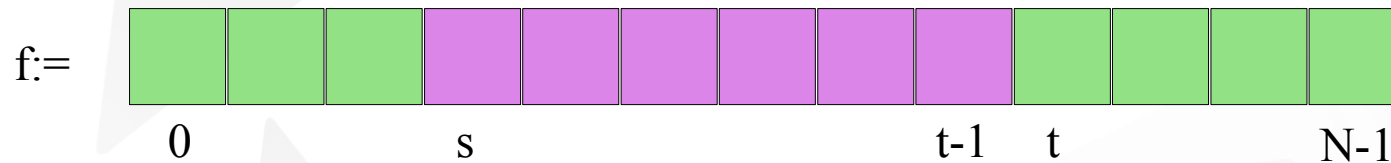
Fila: Implementação em vetor

Uma fila pode ser armazenada em um segmento

$f[s..t-1]$

de um vetor

$f[0..N-1]$



Para inserir o elemento y na fila f ?

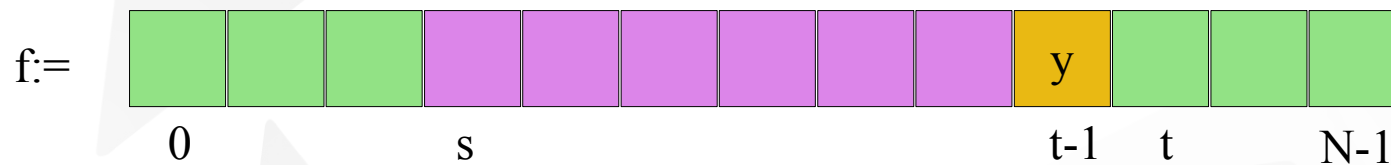
Fila: Implementação em vetor

Uma fila pode ser armazenada em um segmento

$f[s..t-1]$

de um vetor

$f[0..N-1]$



Para inserir o elemento y na fila f :

$f[t] = y;$

$t = t+1$

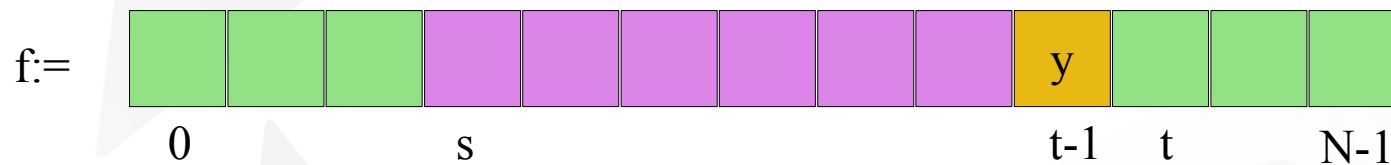
Fila: Implementação em vetor

Uma fila pode ser armazenada em um segmento

$f[s..t-1]$

de um vetor

$f[0..N-1]$



Para inserir o elemento y na fila f :

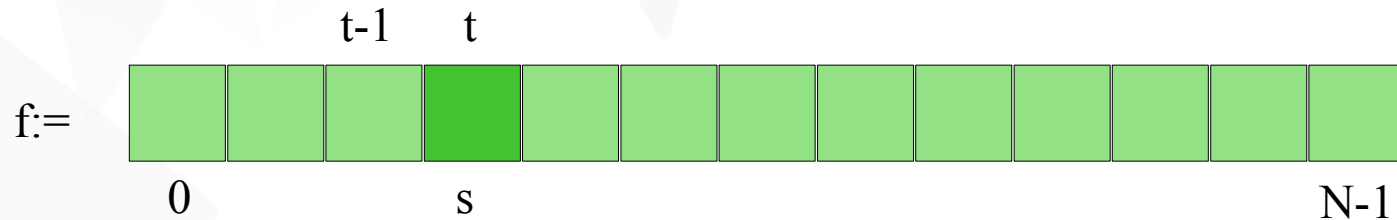
$f[t] = y;$

$t = t+1$

$f[t++] = y;$

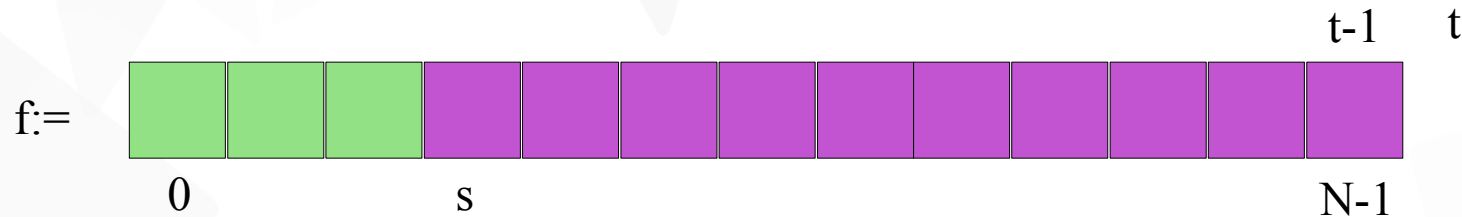
Fila: Alguns cuidados

- **Remover um elemento em uma fila vazia** (quando $s==t$)



Fila: Alguns cuidados

- Inserir um elemento em uma fila cheia ($t == N$)

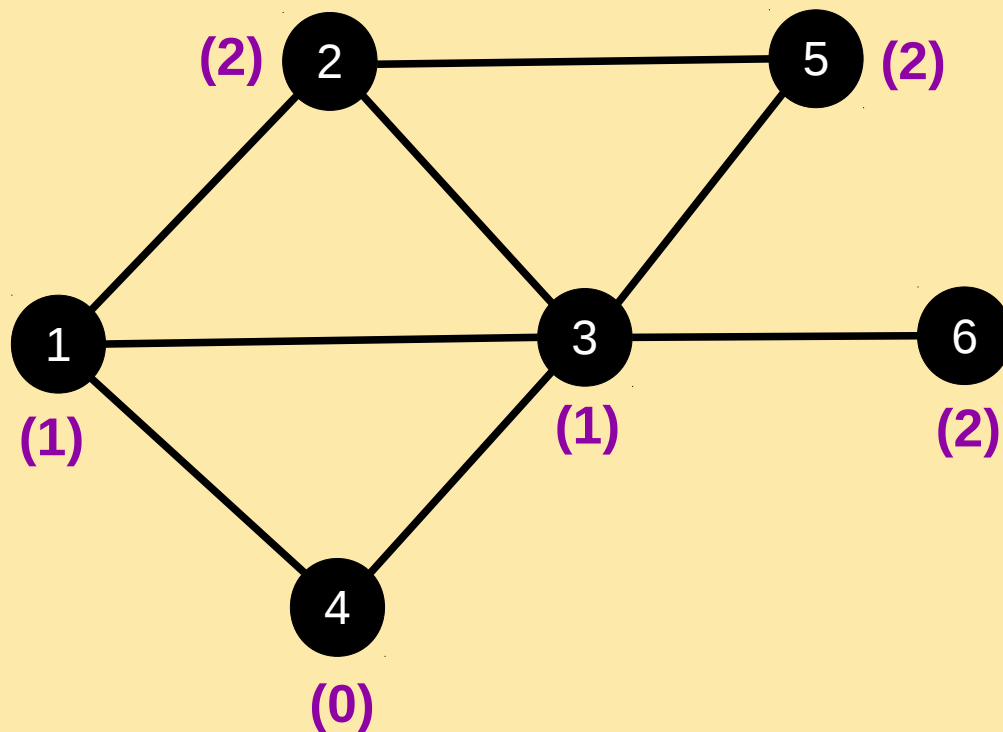


Transbordamento em fila: evento excepcional (mau planejamento lógico)

Distância do vértice 4 aos demais

$c=4$

$d=[1, 2, 1, 0, 2, 2]$



A estrutura de dados
FILA governa a
Varredura na rede


```
// Recebe uma matriz A que representa as interligações entre cidades 0,1,...,5:  
// há uma estrada (de mão única) de x a y se e só se  $A[x][y] == 1$ . Devolve um vetor  
// d tal que d[x] é a distância de c a x para cada x.
```

```
int *distancias (int A[6][6], int c) {  
    int x, y;  
    int fila[6], s, t;  
  
    int *d = malloc(6*sizeof(int));  
  
    for (x=0; x<6; x++)  
        d[x] = 99999;  
  
    d[c] = 0;  
    s = 0;  
    t = 1;  
    fila[0] = c; // c entra na fila  
  
    while (s<t) {  
        x = fila[s++]; // x sai da fila  
  
        for (y=0; y<6; y++) {  
            if (A[x][y]==1 && d[y]==99999) {  
                d[y] = d[x] + 1;  
                fila[t++] = y; // y entra na fila  
            }  
        }  
    }  
  
    return d;  
}
```

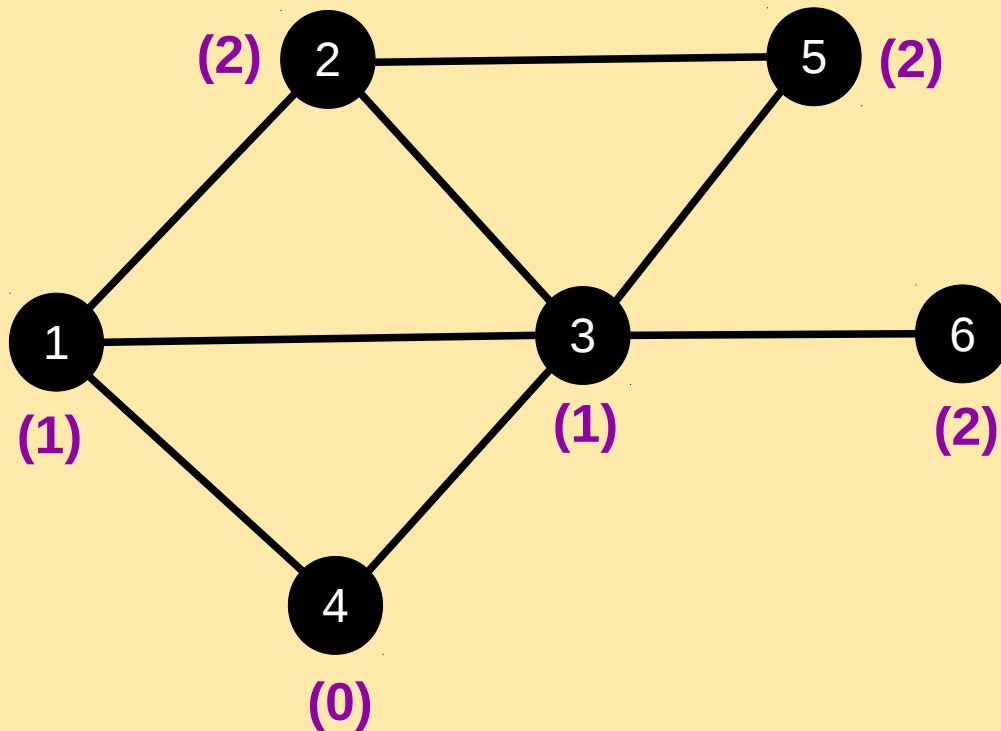
```
// Recebe uma matriz A que representa as interligações entre cidades 0,1,...,5:  
// há uma estrada (de mão única) de x a y se e só se  $A[x][y] == 1$ . Devolve um vetor  
// d tal que d[x] é a distância de c a x para cada x.
```

```
int *distancias (int A[6][6], int c) {  
    int x, y;  
    int fila[6], s, t;  
  
    int *d = malloc(6*sizeof(int));  
  
    for (x=0; x<6; x++)  
        d[x] = 99999;  
  
    d[c] = 0;  
    s = 0;  
    t = 1;  
    fila[0] = c; // c entra na fila  
  
    while (s<t) {  
        x = fila[s++]; // x sai da fila  
  
        for (y=0; y<6; y++) {  
            if (A[x][y]==1 && d[y]==99999) {  
                d[y] = d[x] + 1;  
                fila[t++] = y; // y entra na fila  
            }  
        }  
    }  
  
    return d;  
}
```

Complexidade computacional?

$c=4$

$d=[1, 2, 1, 0, 2, 2]$



$O(|V|+|E|)$

```
// Recebe uma matriz A que representa as interligações entre cidades 0,1,...,5:  
// há uma estrada (de mão única) de x a y se e só se  $A[x][y] == 1$ . Devolve um vetor  
// d tal que d[x] é a distância de c a x para cada x.
```

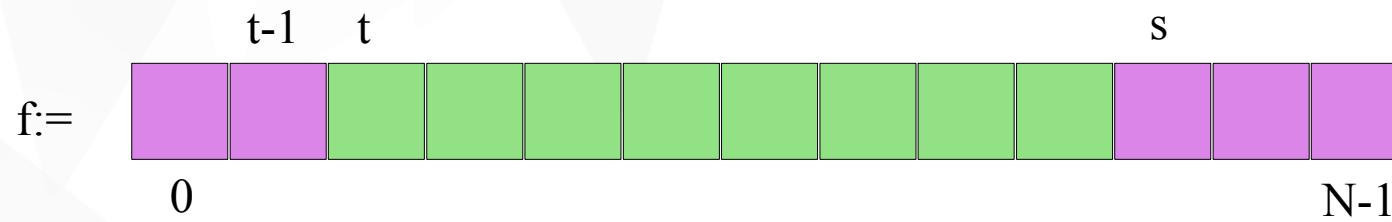
```
int *distancias (int A[6][6], int c) {  
    int x, y;  
    int fila[6], s, t;  
  
    int *d = malloc(6*sizeof(int));  
  
    for (x=0; x<6; x++)  
        d[x] = 99999;  
  
    d[c] = 0;  
    s = 0;  
    t = 1;  
    fila[0] = c; // c entra na fila  
  
    while (s<t) {  
        x = fila[s++]; // x sai da fila  
  
        for (y=0; y<6; y++) {  
            if (A[x][y]==1 && d[y]==99999) {  
                d[y] = d[x] + 1;  
                fila[t++] = y; // y entra na fila  
            }  
        }  
    }  
  
    return d;  
}
```

Em outras linguagens de programação

- Inserir (colocar): **enqueue**
- Remover (retirar): **de-queue**

Na aula de prática:

- Fila circular



- Pilha

