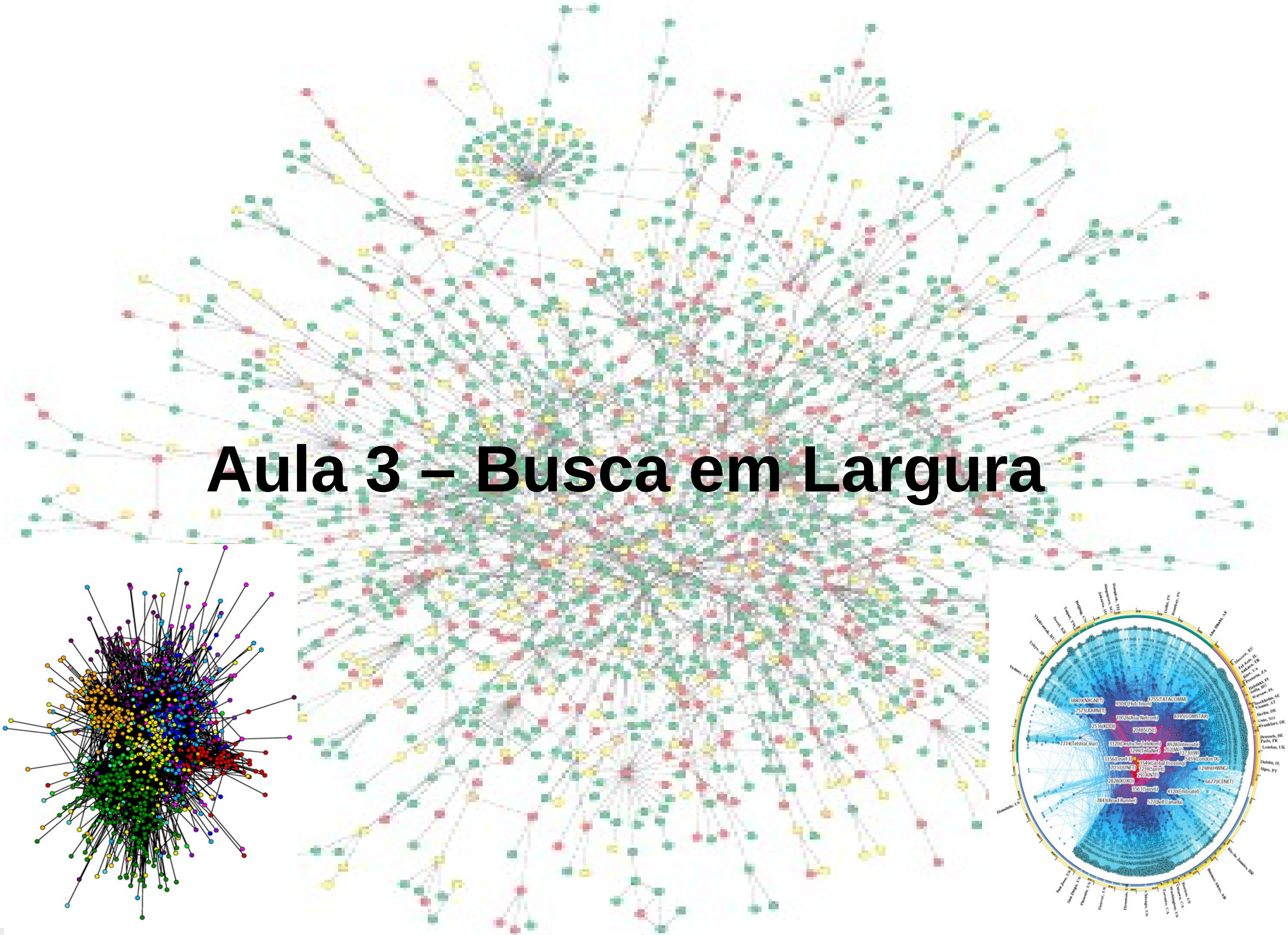
Aula 3 – Busca em Largura



Aulas passadas

- Problema das 7 pontes de Konigsberg
- Definições e propriedades
- Representação de grafos.

Roteiro da aula

- **Algoritmos de buscas em grafos**
  - Busca em largura
- **Atividade prática**

II. Buscas em grafos
(algoritmos de varredura em grafos)

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

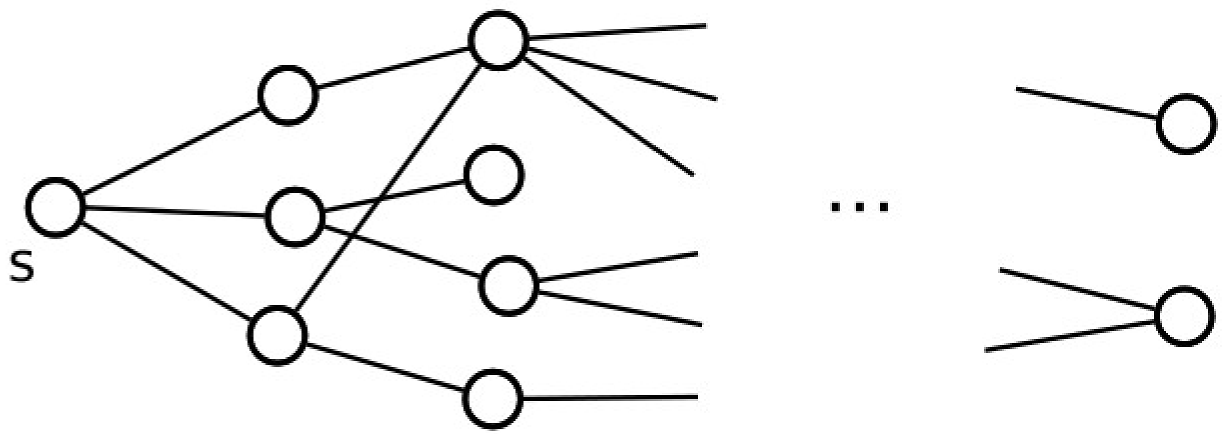
- Os algoritmos de busca em grafos permite percorrer o grafo **buscando todos os vértices que são acessíveis a partir de um determinado vértice em questão**.

- 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.

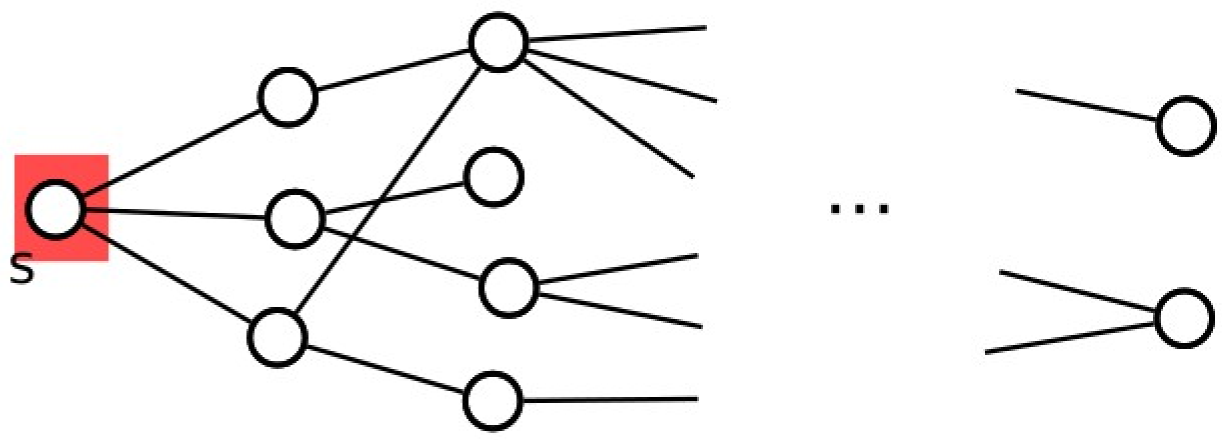
II. Buscas em grafos
(algoritmos de varredura em grafos)

Busca em largura



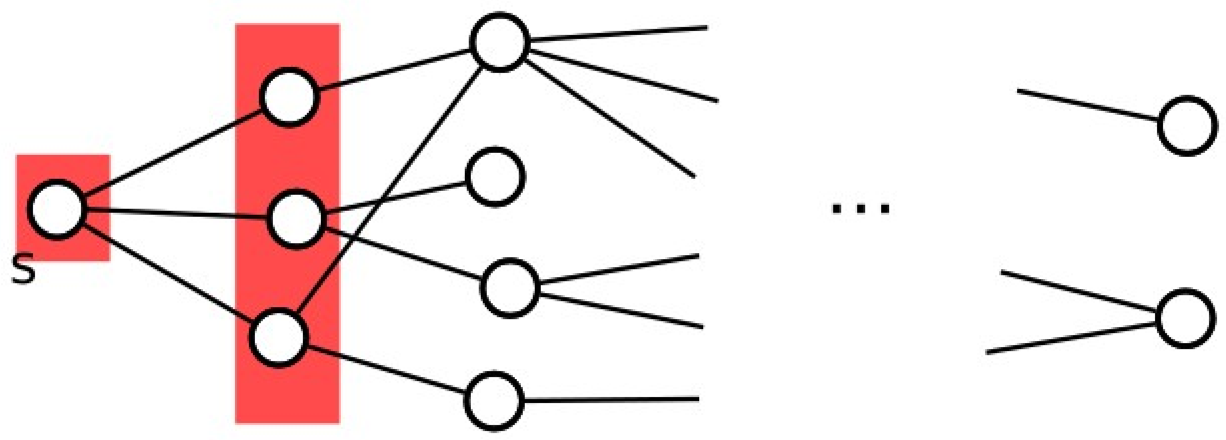
- Inicialmente **{s}**.
- Os níveis são explorados “geologicamente”.
- Fronteira = nível atual.
- Iterativamente avançar a fronteira para o nível seguinte, cuidando para não voltar ao nível inferior.

Busca em largura



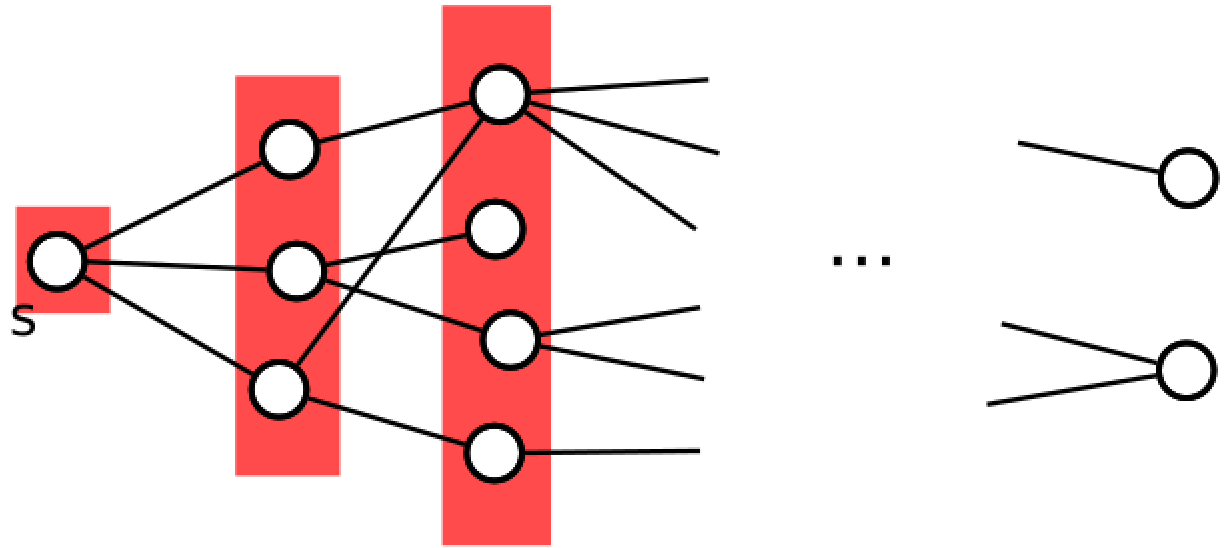
- Inicialmente **{s}**.
- Os níveis são explorados “geologicamente”.
- Fronteira = nível atual.
- Iterativamente avançar a fronteira para o nível seguinte, cuidando para não voltar ao nível inferior.

Busca em largura



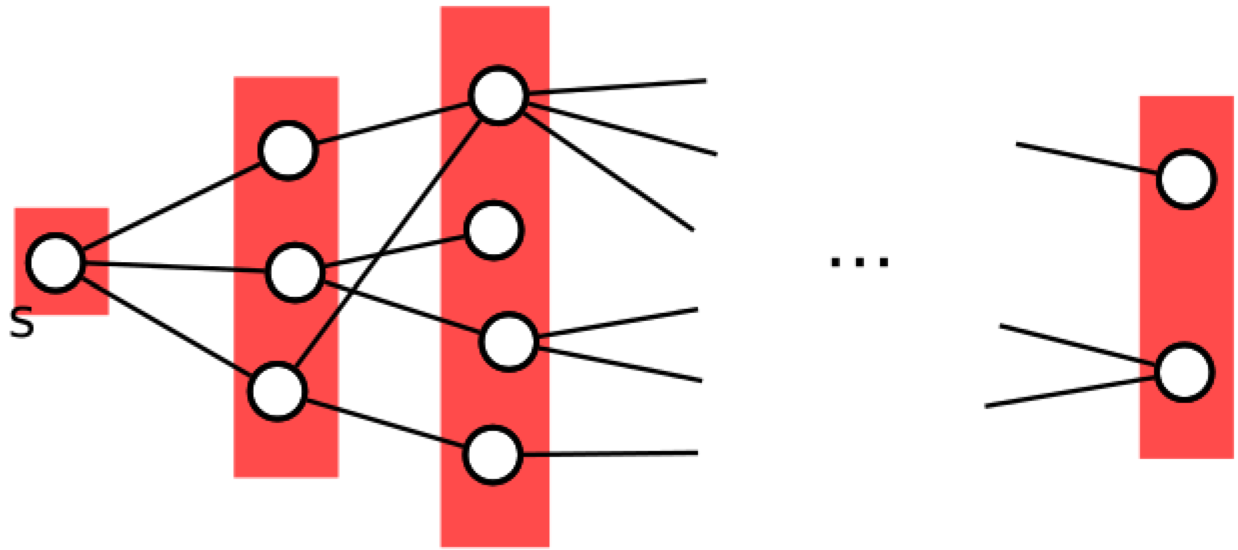
- Inicialmente **{s}**.
- Os níveis são explorados “geologicamente”.
- Fronteira = nível atual.
- Iterativamente avançar a fronteira para o nível seguinte, cuidando para não voltar ao nível inferior.

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.

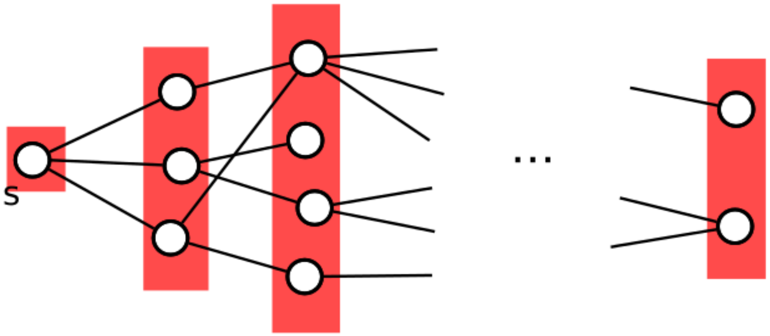
Busca em largura

- 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**.

- Constrói-se 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)*

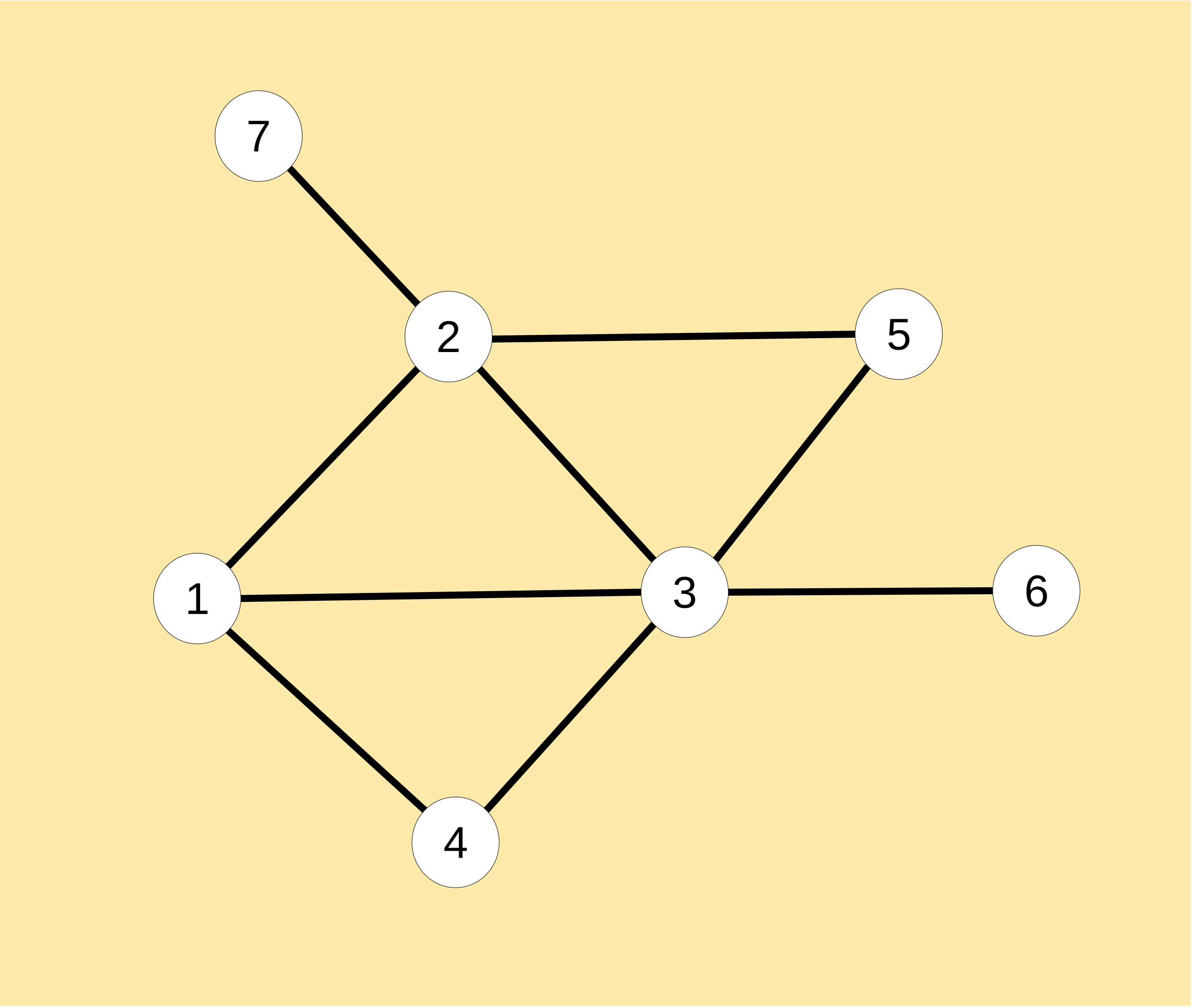
Busca em largura

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".

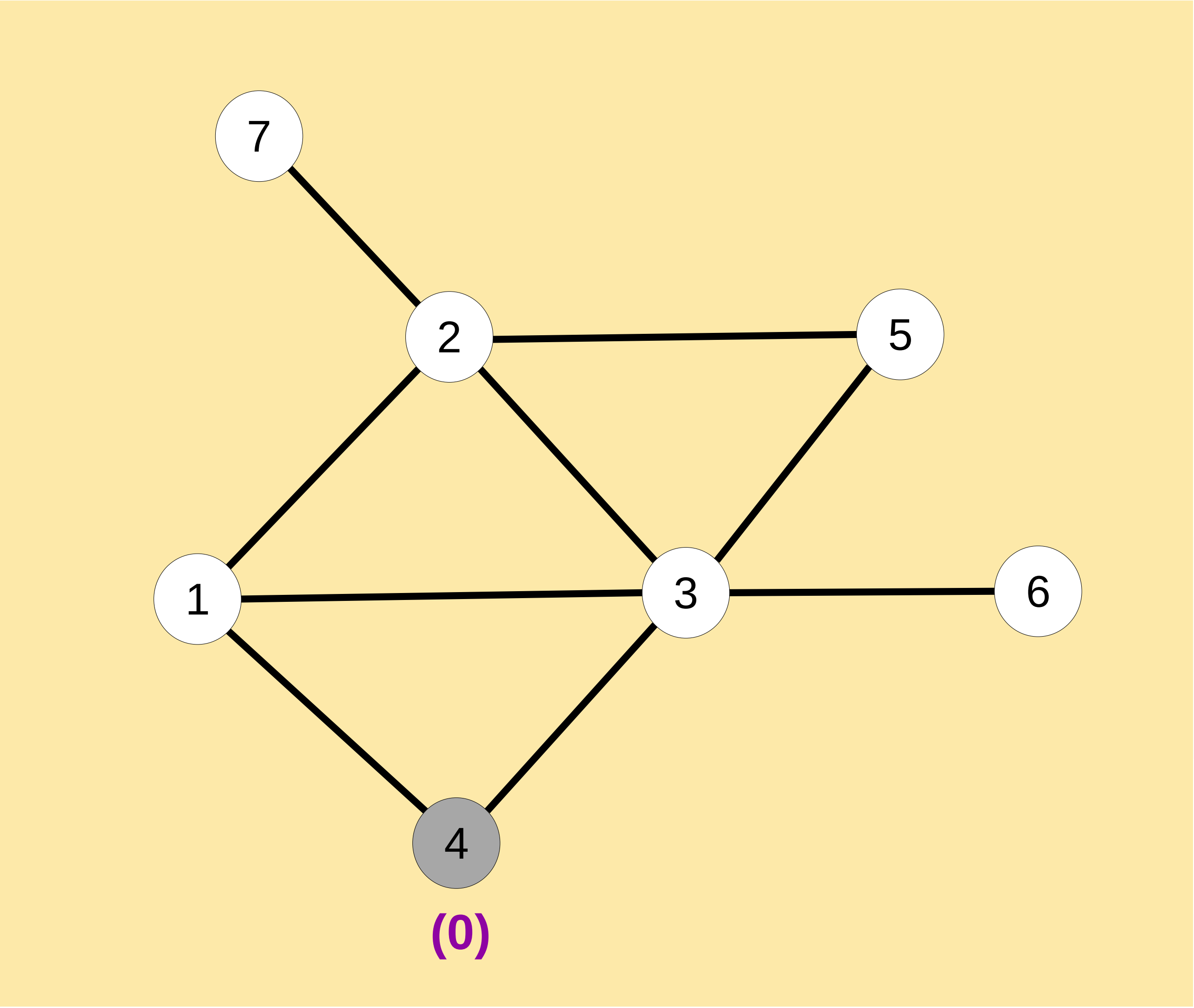
Busca em largura

Grafo inicial



Busca em largura (origem s=4)

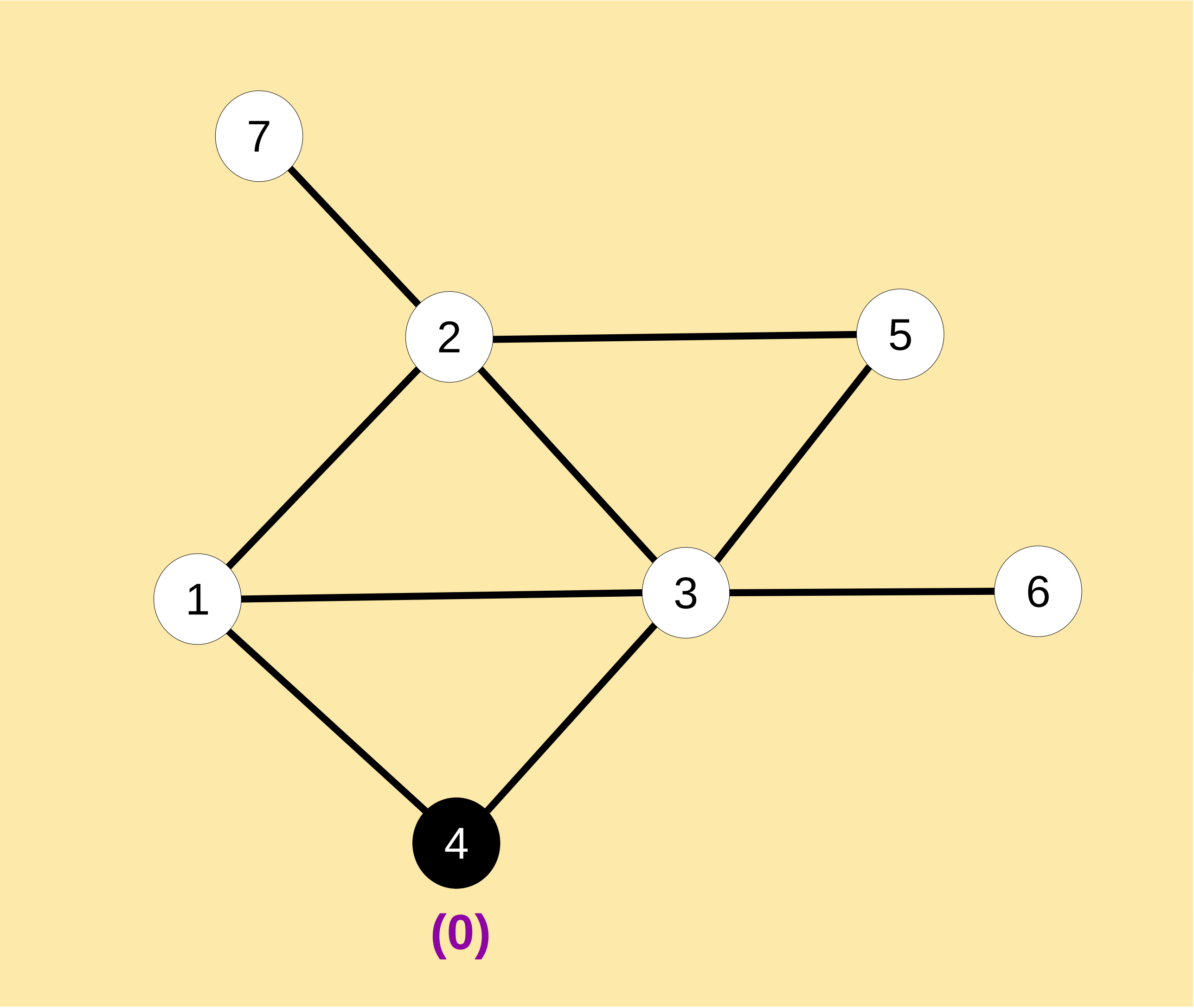
Passo 0



Fila = {4}

Busca em largura (origem s=4)

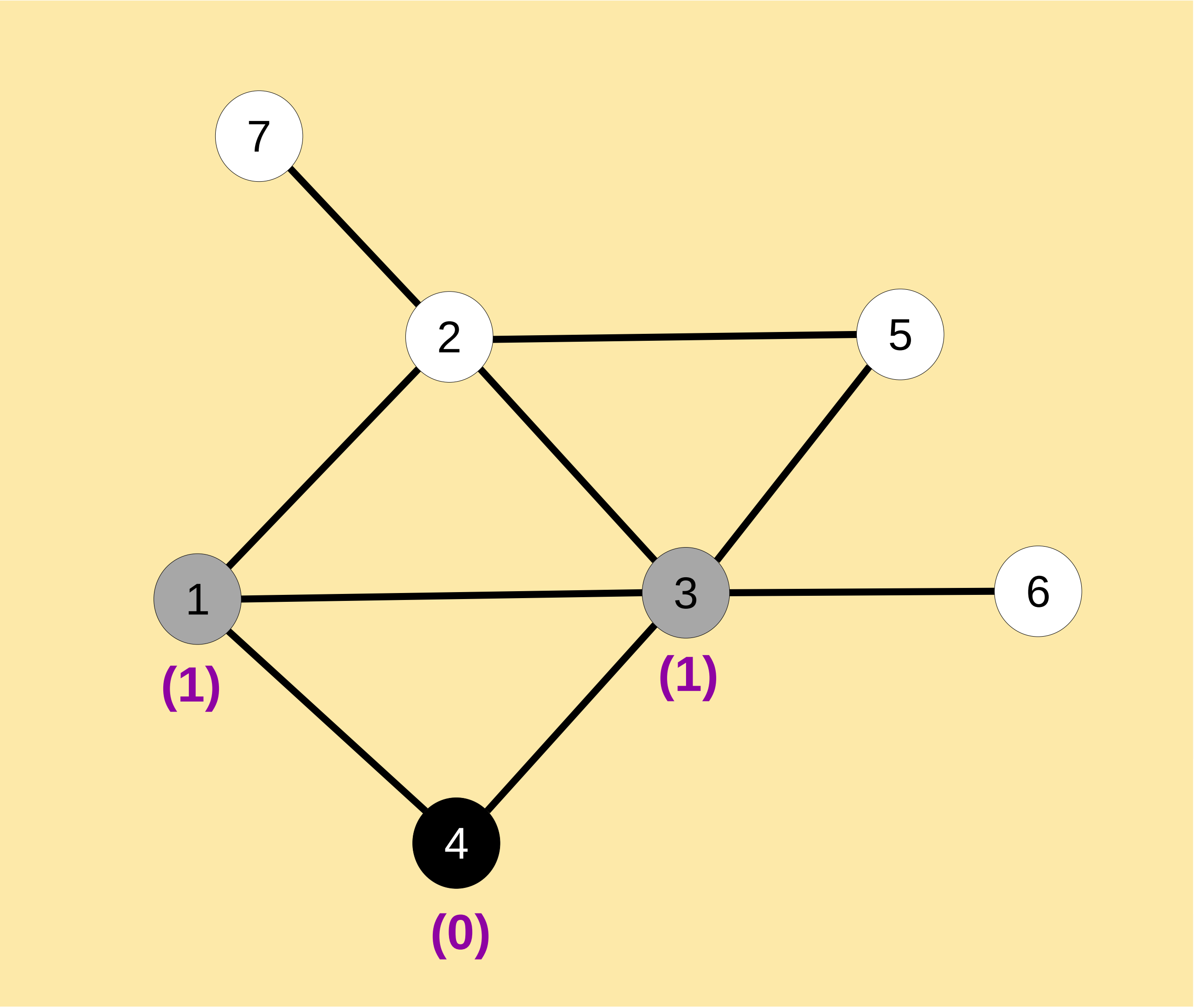
Passo 1



Fila = {}

Busca em largura (origem s=4)

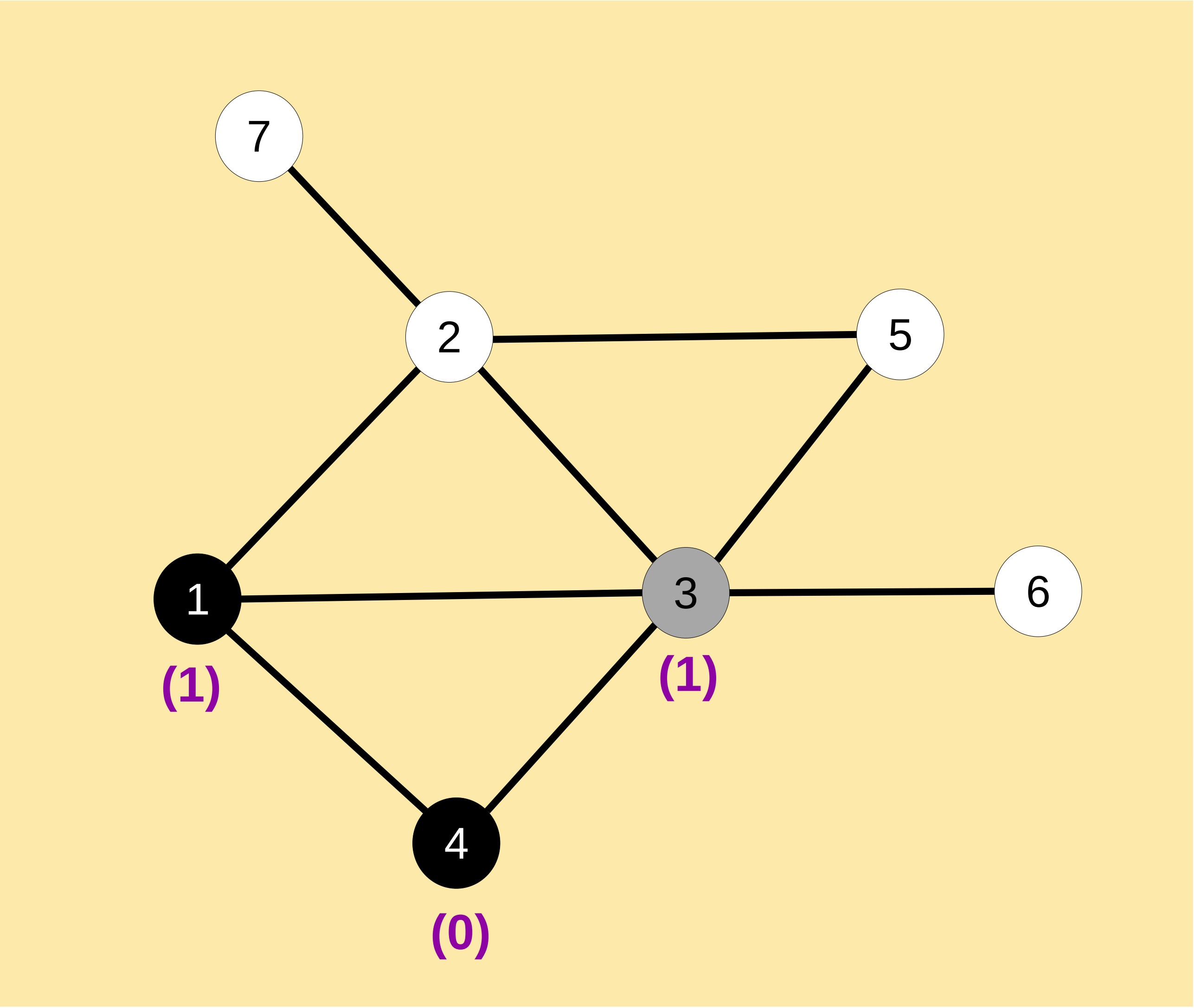
Passo 2



Fila = {1,3}

Busca em largura (origem s=4)

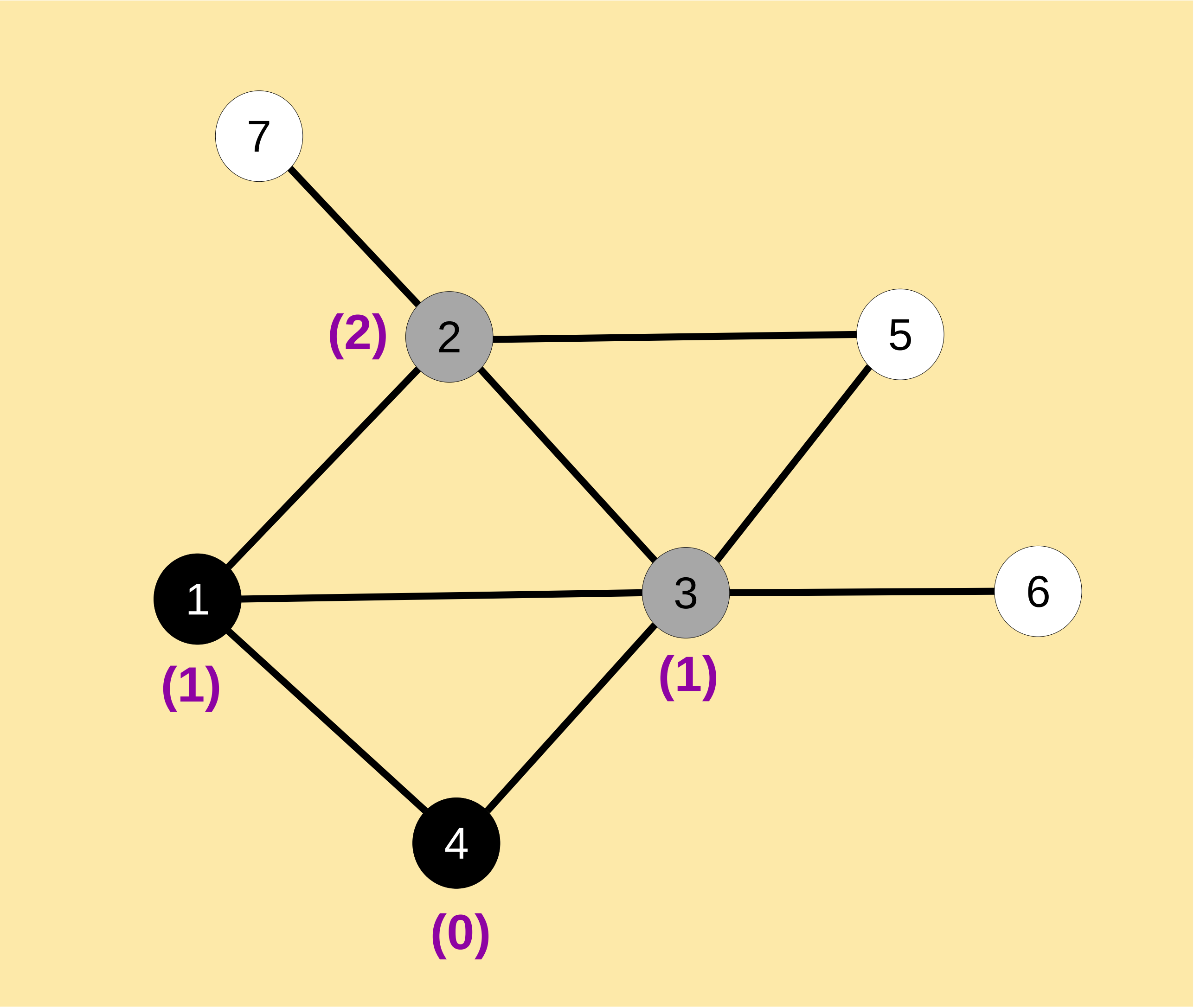
Passo 3



Fila = {3}

Busca em largura (origem s=4)

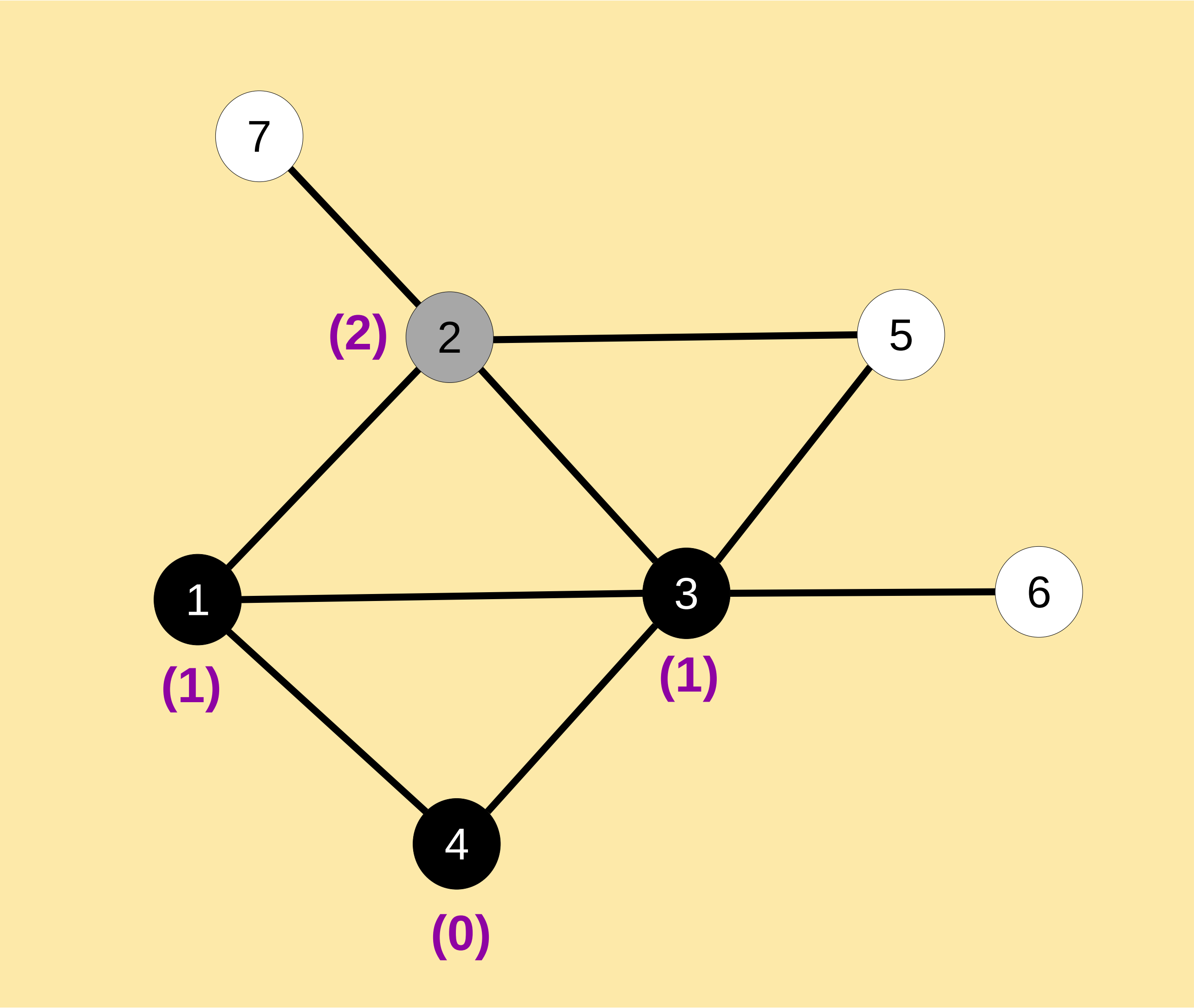
Passo 4



Fila = {3,2}

Busca em largura (origem s=4)

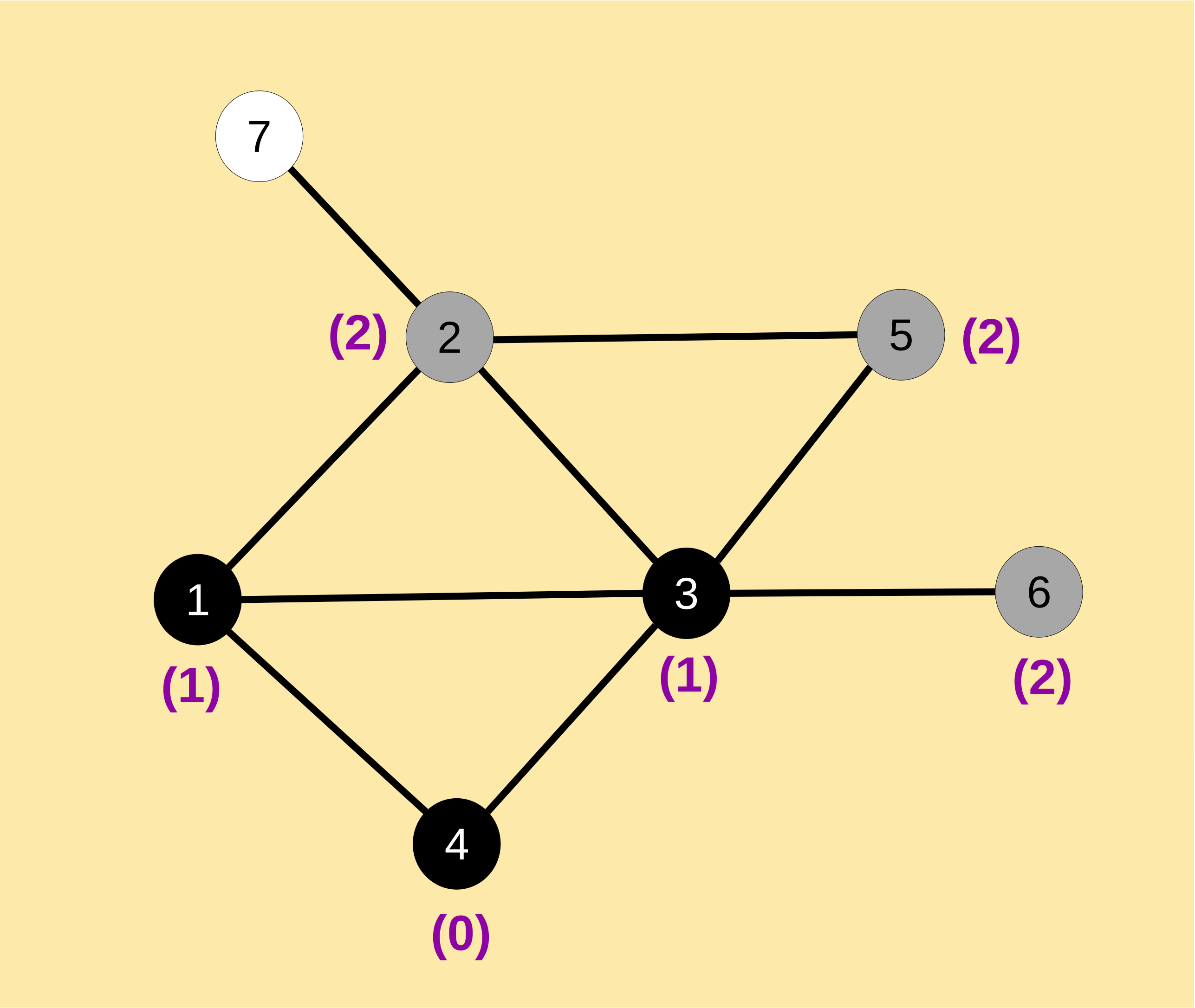
Passo 5



Fila = {2}

Busca em largura (origem s=4)

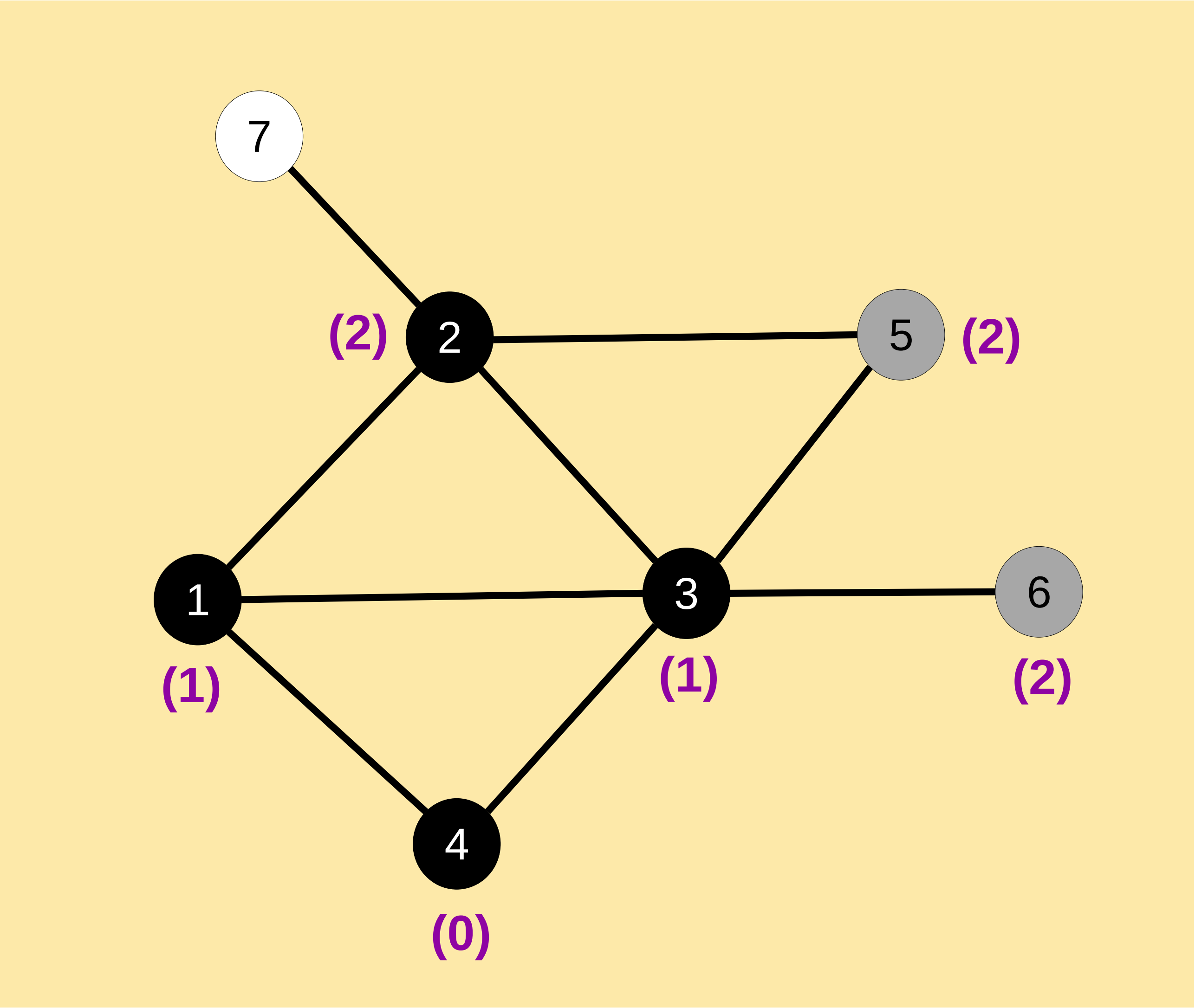
Passo 6



Fila = {2,6,5}

Busca em largura (origem s=4)

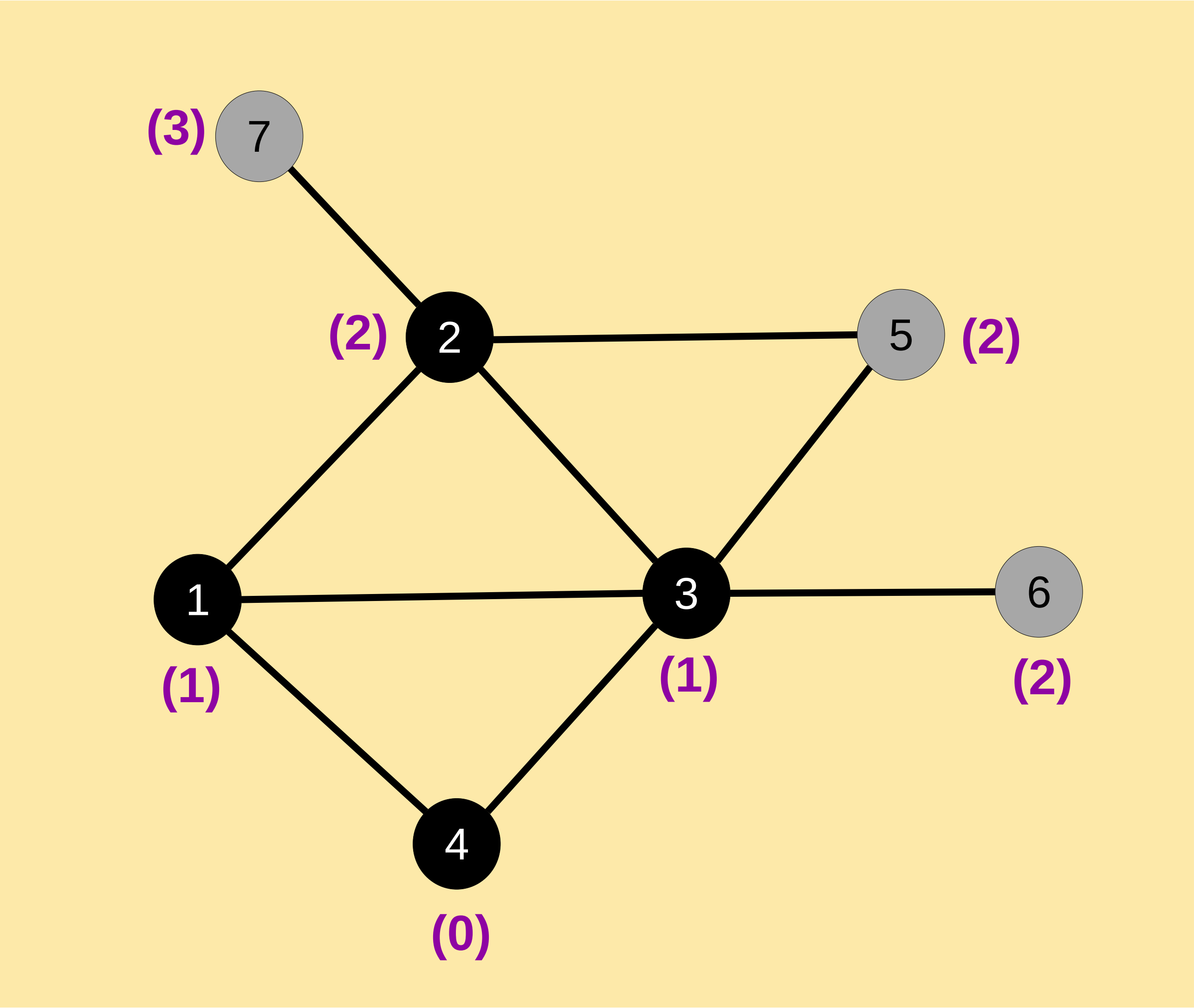
Passo 7



Fila = {6,5}

Busca em largura (origem s=4)

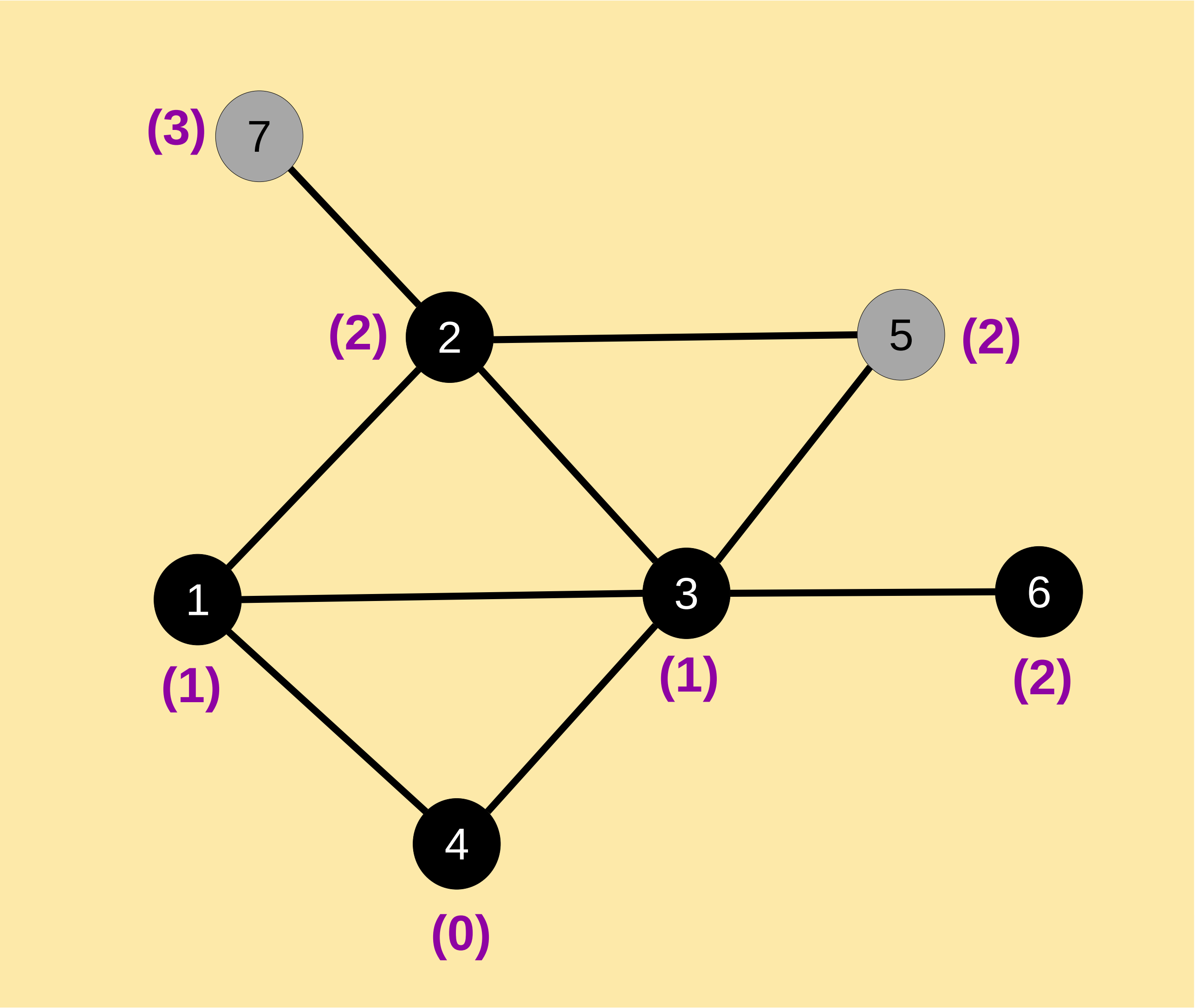
Passo 8



Fila = {6,5,7}

Busca em largura (origem s=4)

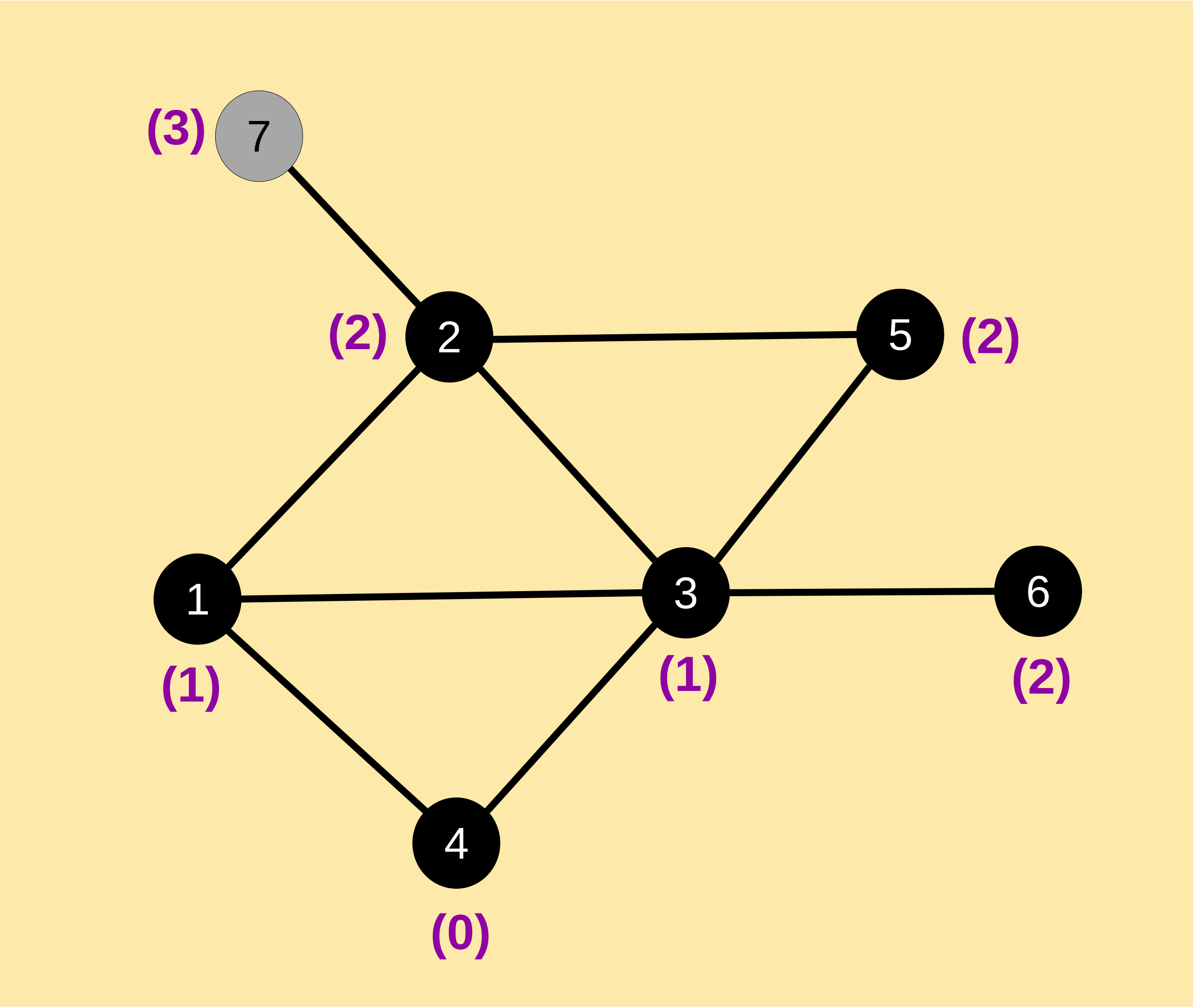
Passo 9



Fila = {5,7}

Busca em largura (origem s=4)

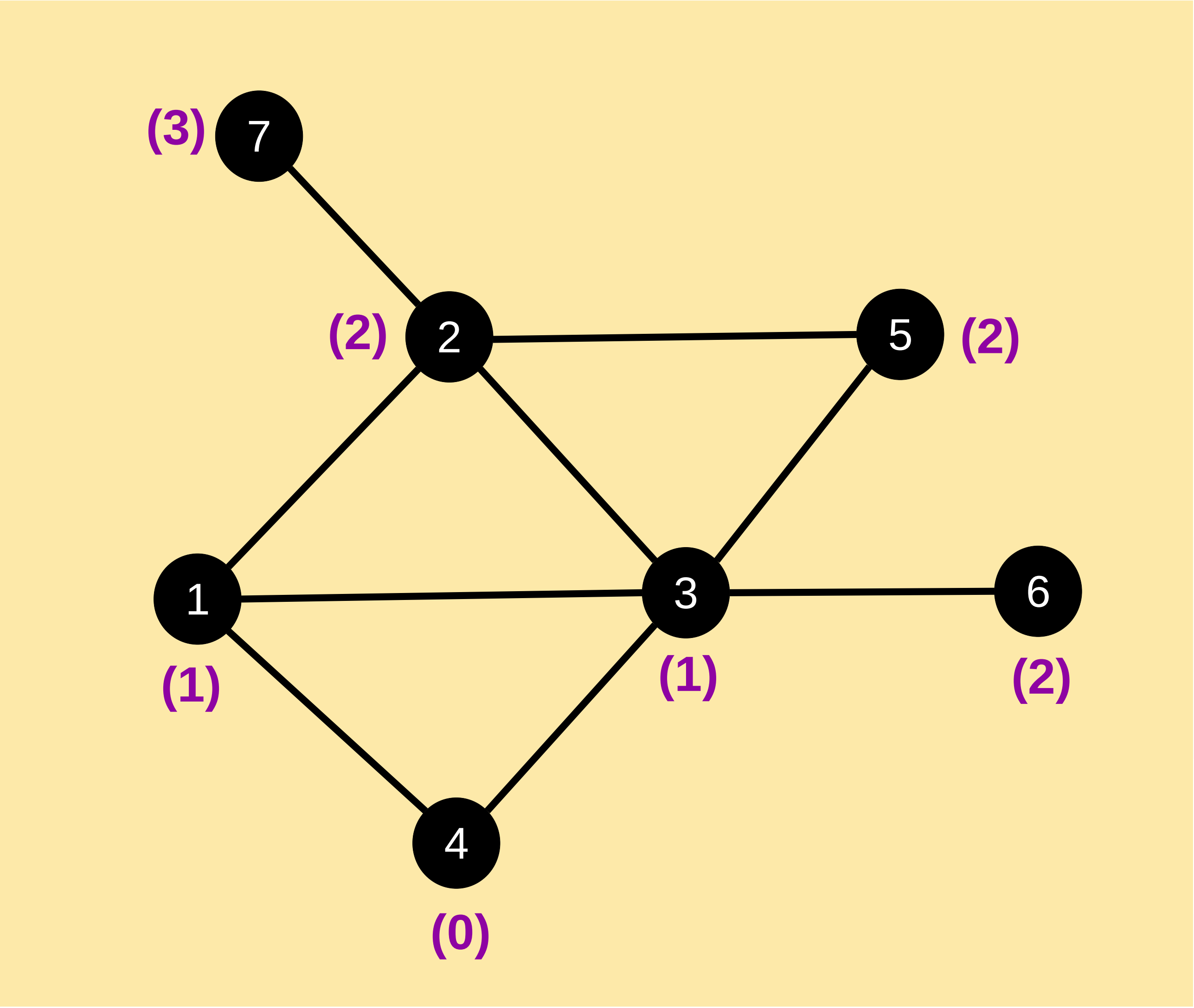
Passo 10



Fila = {7}

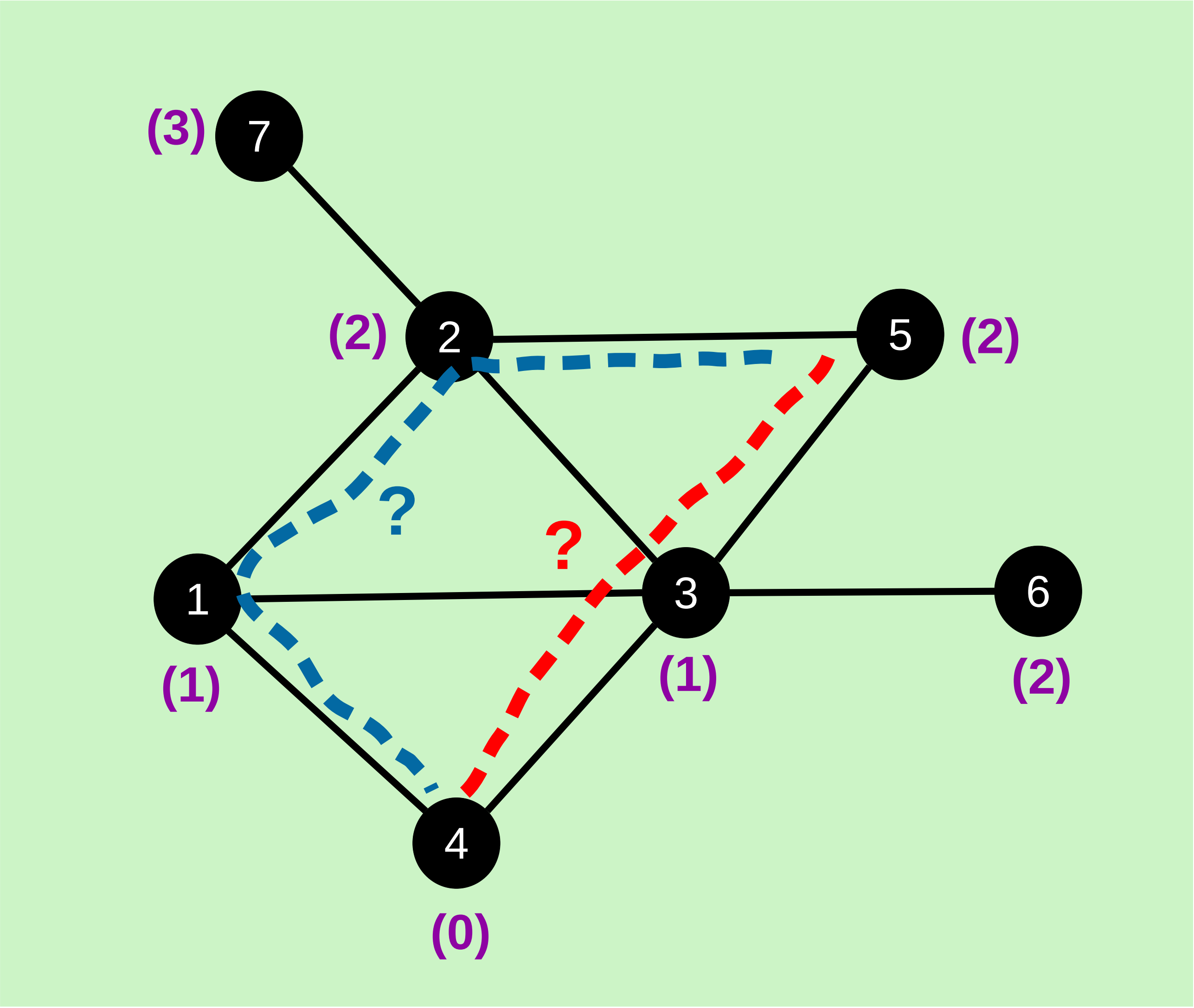
Busca em largura (origem s=4)

Passo 11



Fila = {}

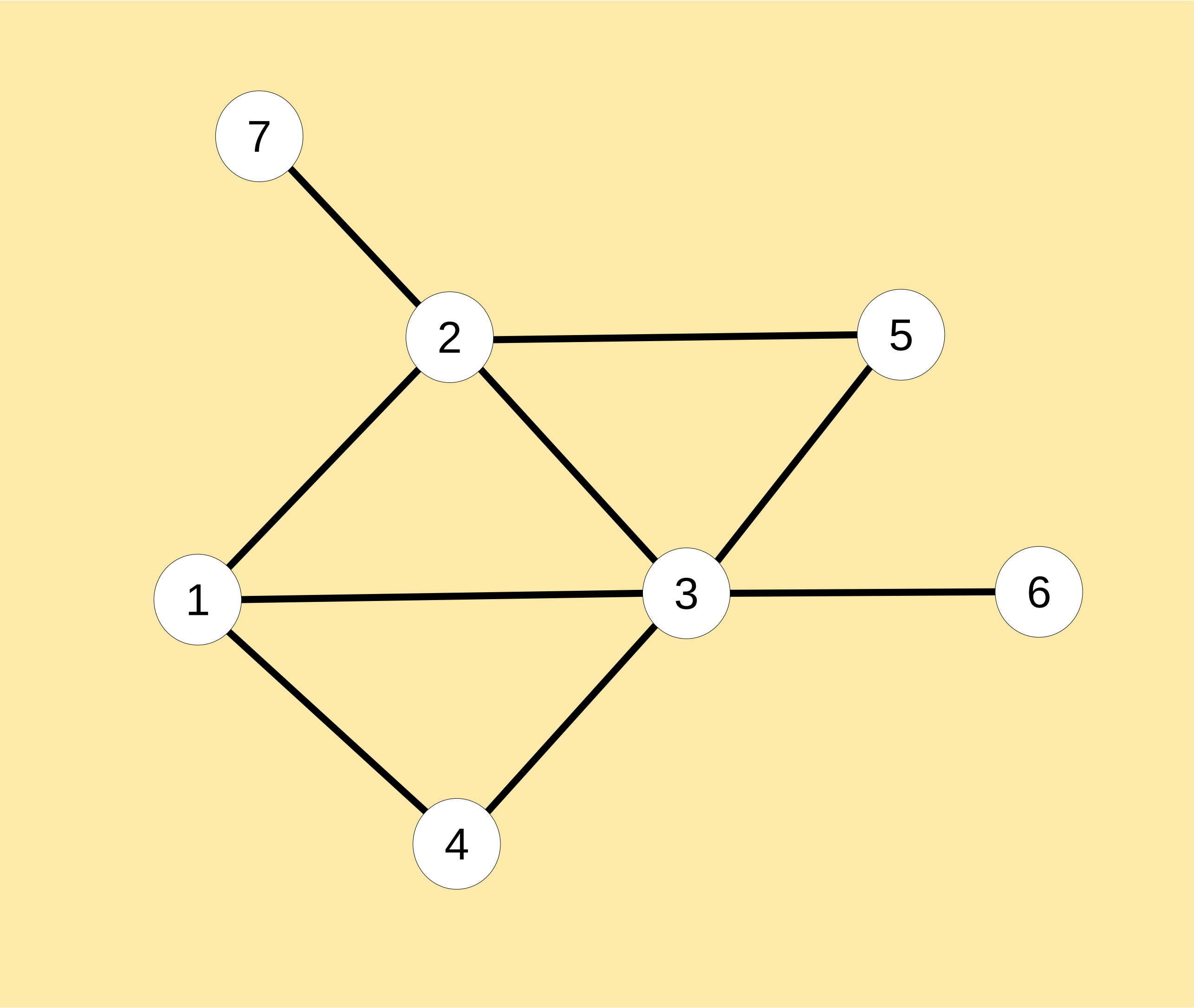
Busca em largura



Como determinamos o caminho exato para: a partir do vértice **4** chegar ao vértice **5**?

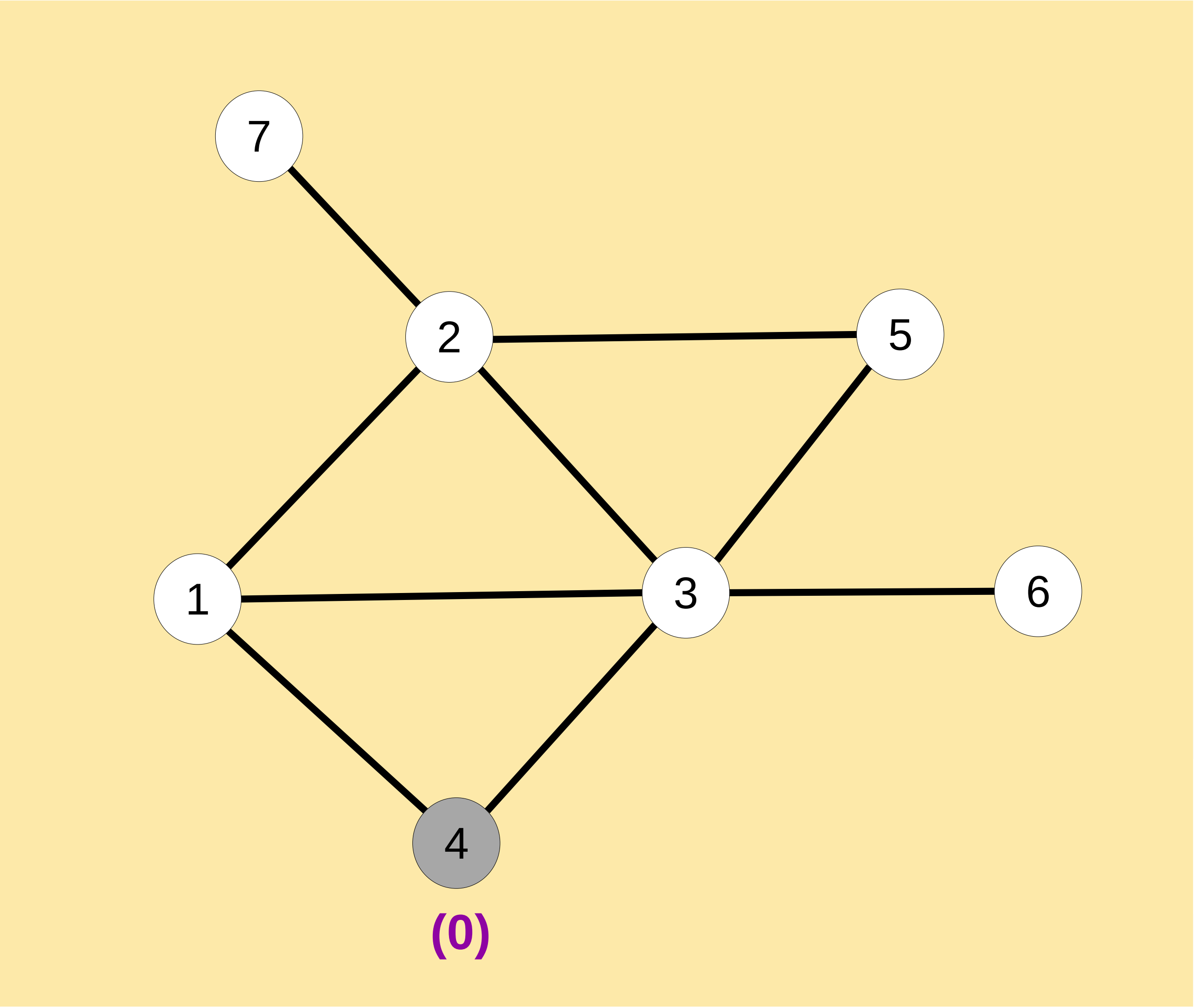
Busca em largura

Grafo inicial



Busca em largura (origem s=4)

Passo 0

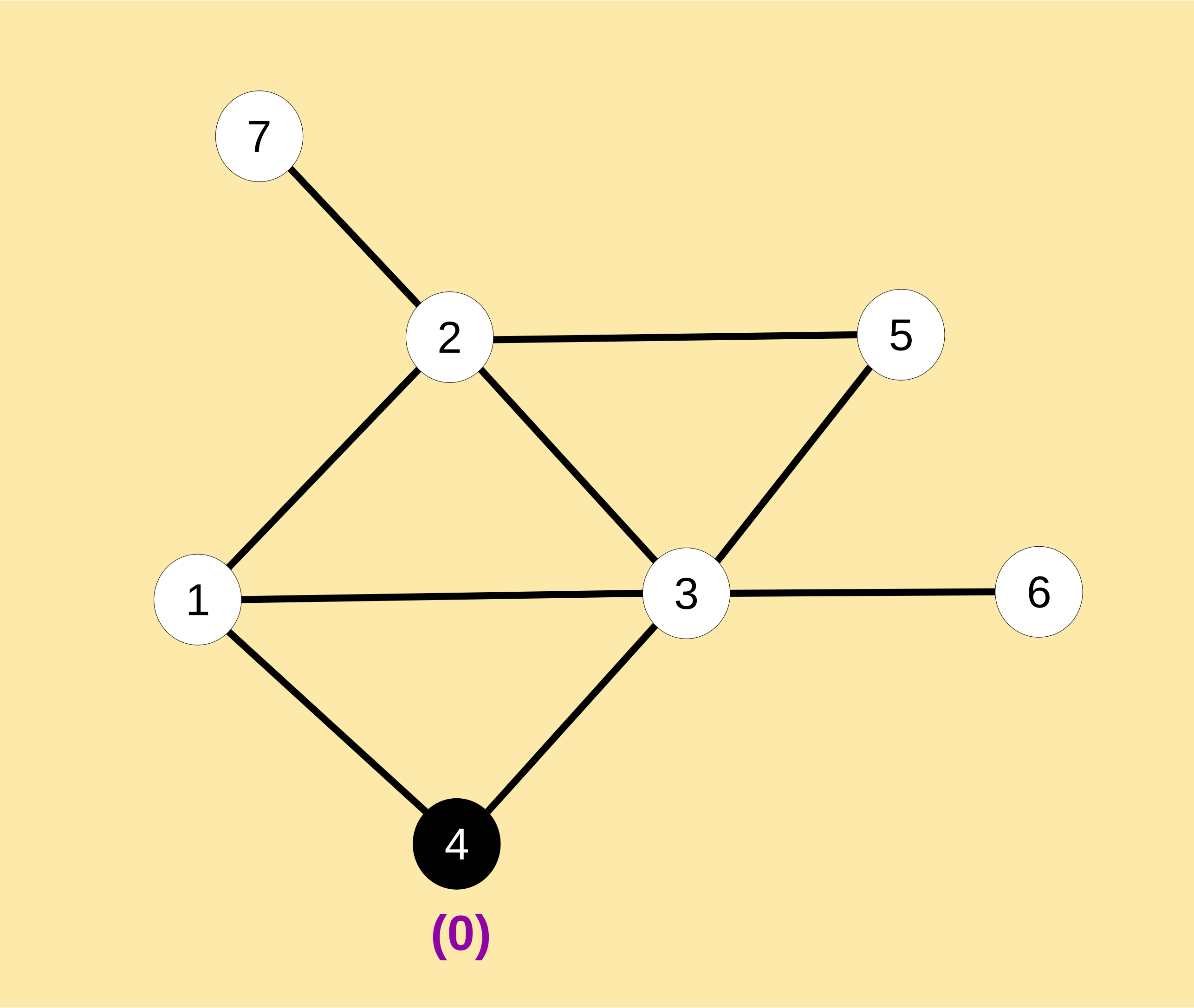


Fila = {4}

Predecessores:
1:
2:
3:
4: --
5:
6:
7:

Busca em largura (origem s=4)

Passo 1

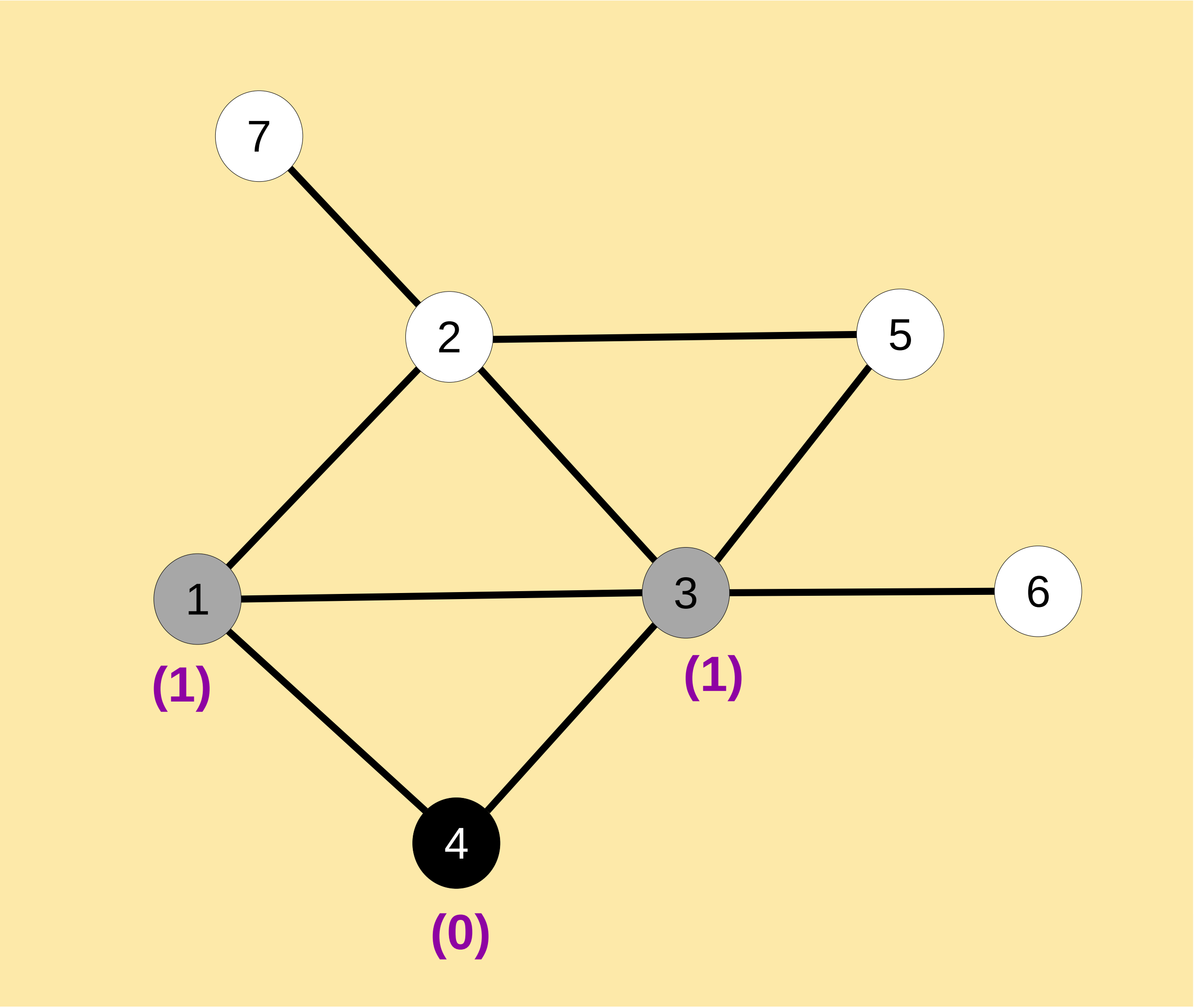


Fila = {}

Predecessores:
1:
2:
3:
4: --
5:
6:
7:

Busca em largura (origem s=4)

Passo 2

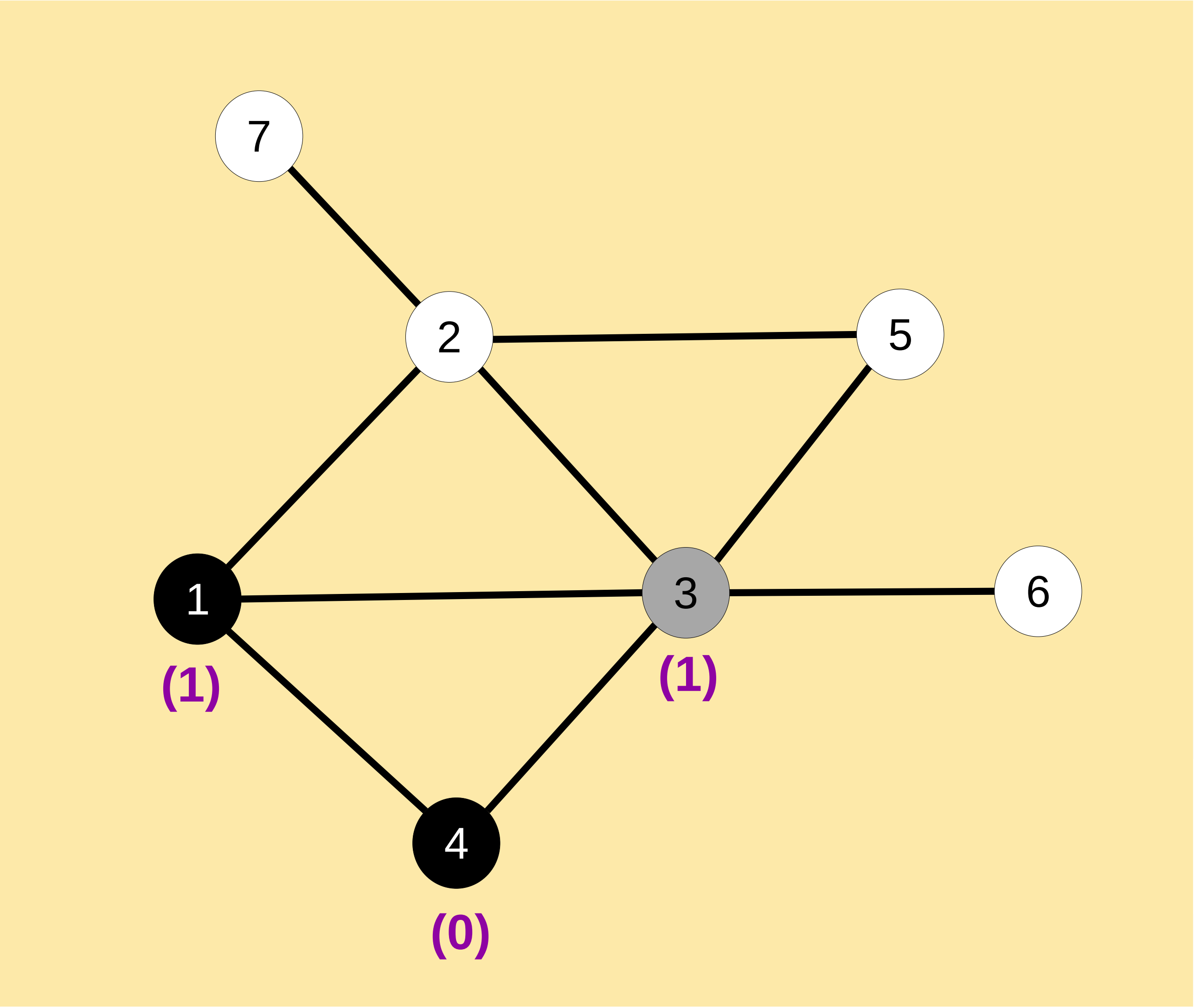


Fila = {1,3}

Predecessores:
1: 4
2:
3: 4
4: --
5:
6:
7:

Busca em largura (origem s=4)

Passo 3

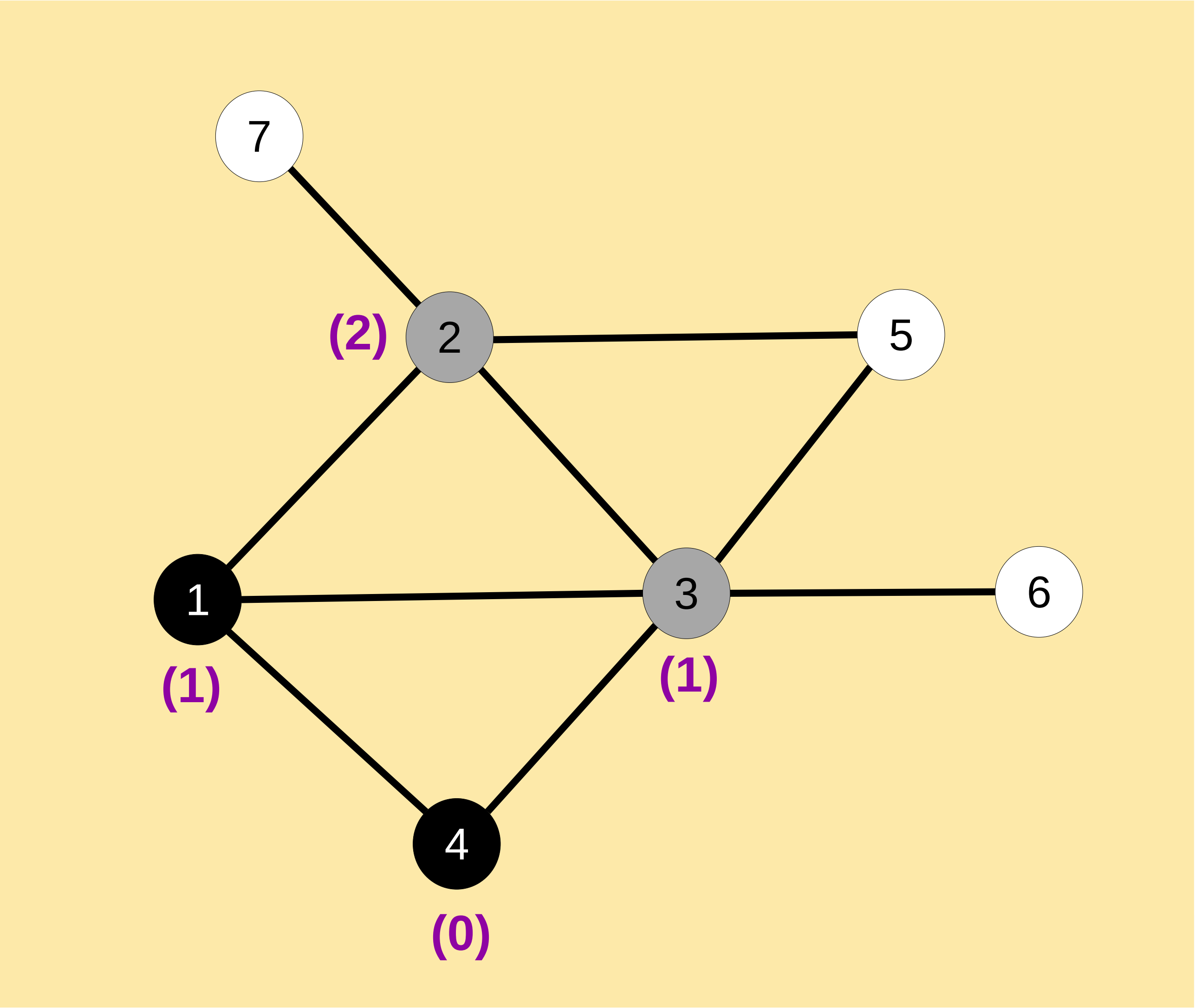


Fila = {3}

Predecessores:
1: 4
2:
3: 4
4: --
5:
6:
7:

Busca em largura (origem s=4)

Passo 4

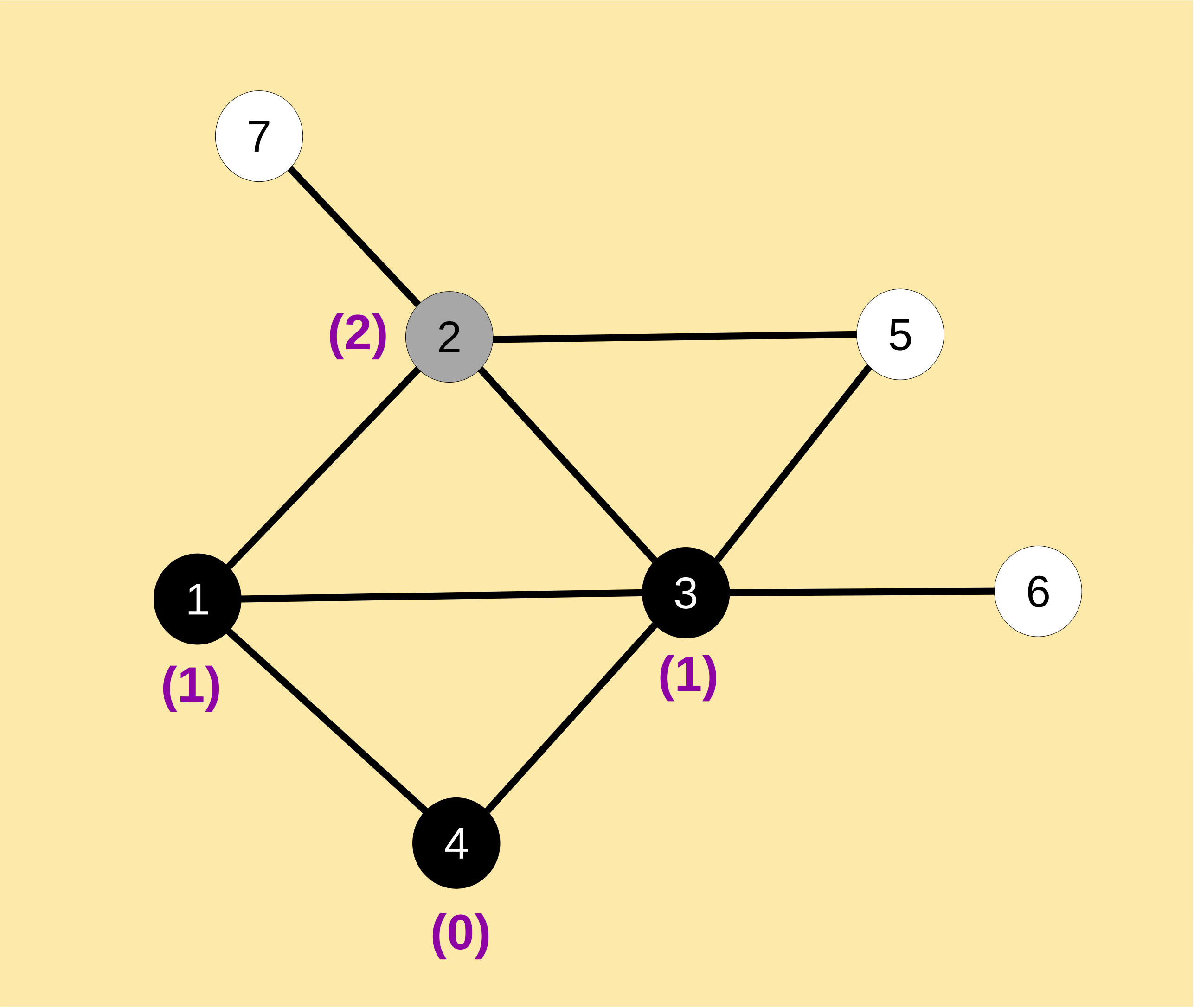


Fila = {3,2}

Predecessores:
1: 4
2: 1
3: 4
4: --
5:
6:
7:

Busca em largura (origem s=4)

Passo 5

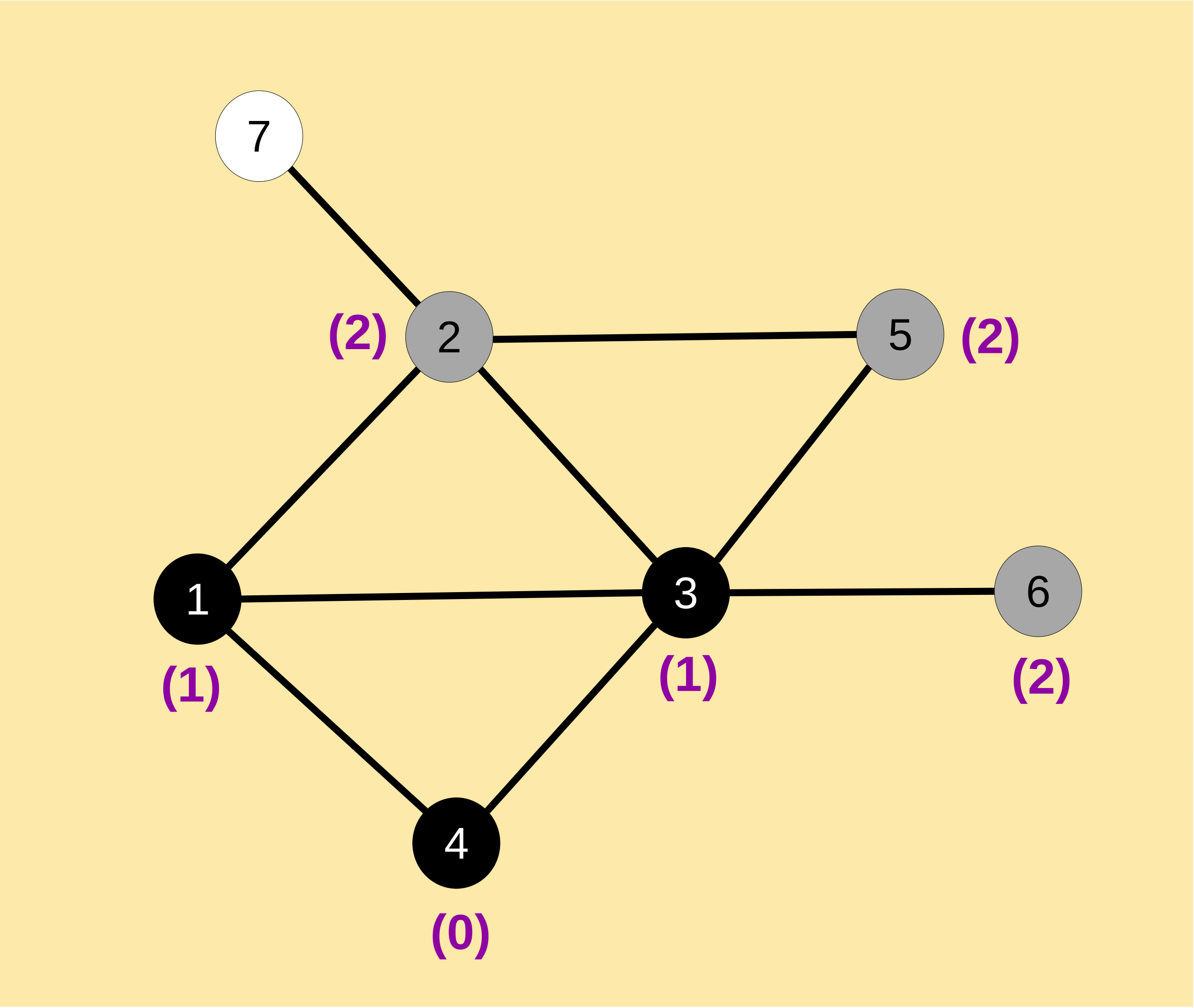


Fila = {2}

Predecessores:
1: 4
2: 1
3: 4
4: --
5:
6:
7:

Busca em largura (origem s=4)

Passo 6

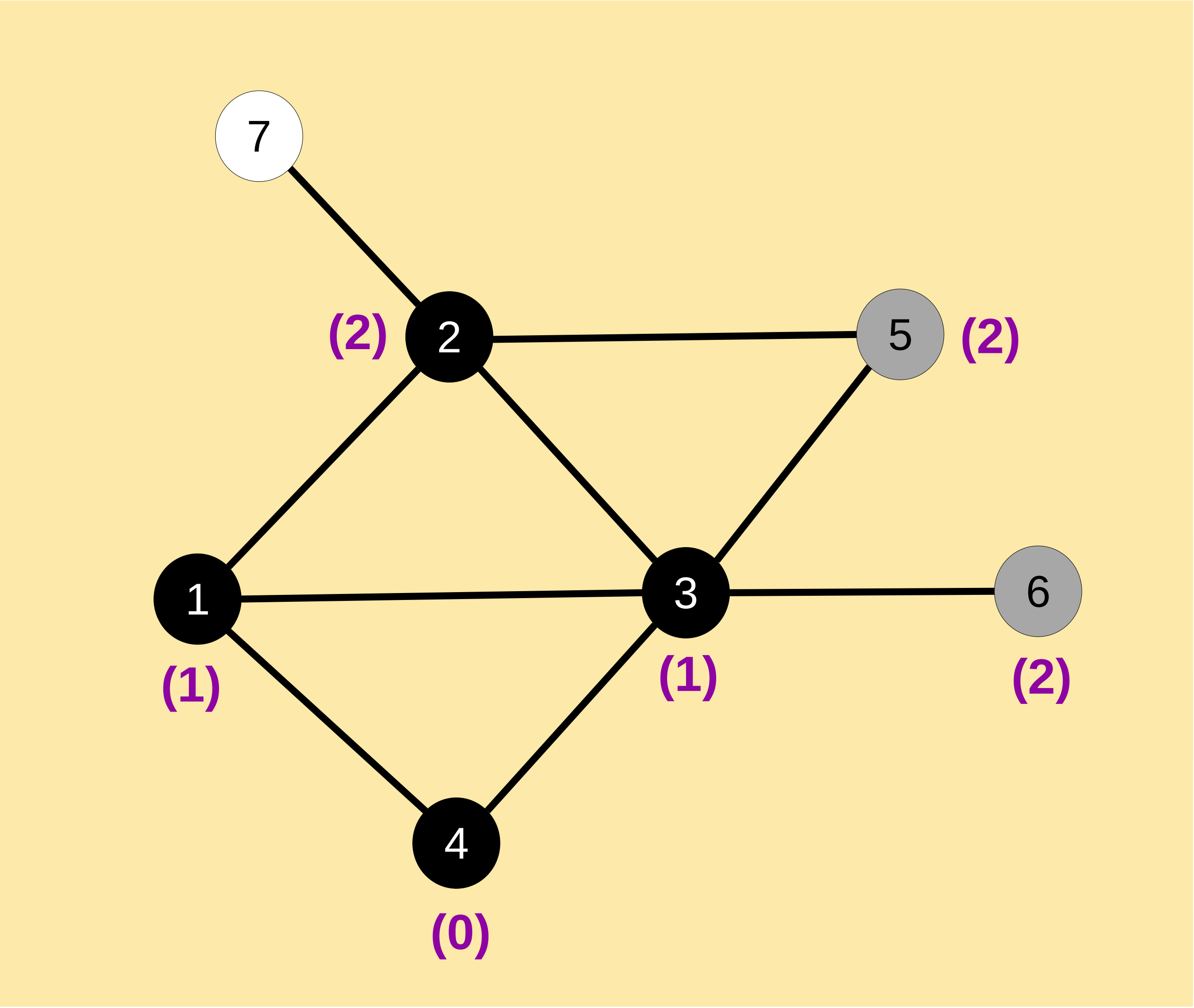


Fila = {2,6,5}

Predecessores:
1: 4
2: 1
3: 4
4: --
5: 3
6: 3
7:

Busca em largura (origem s=4)

Passo 7

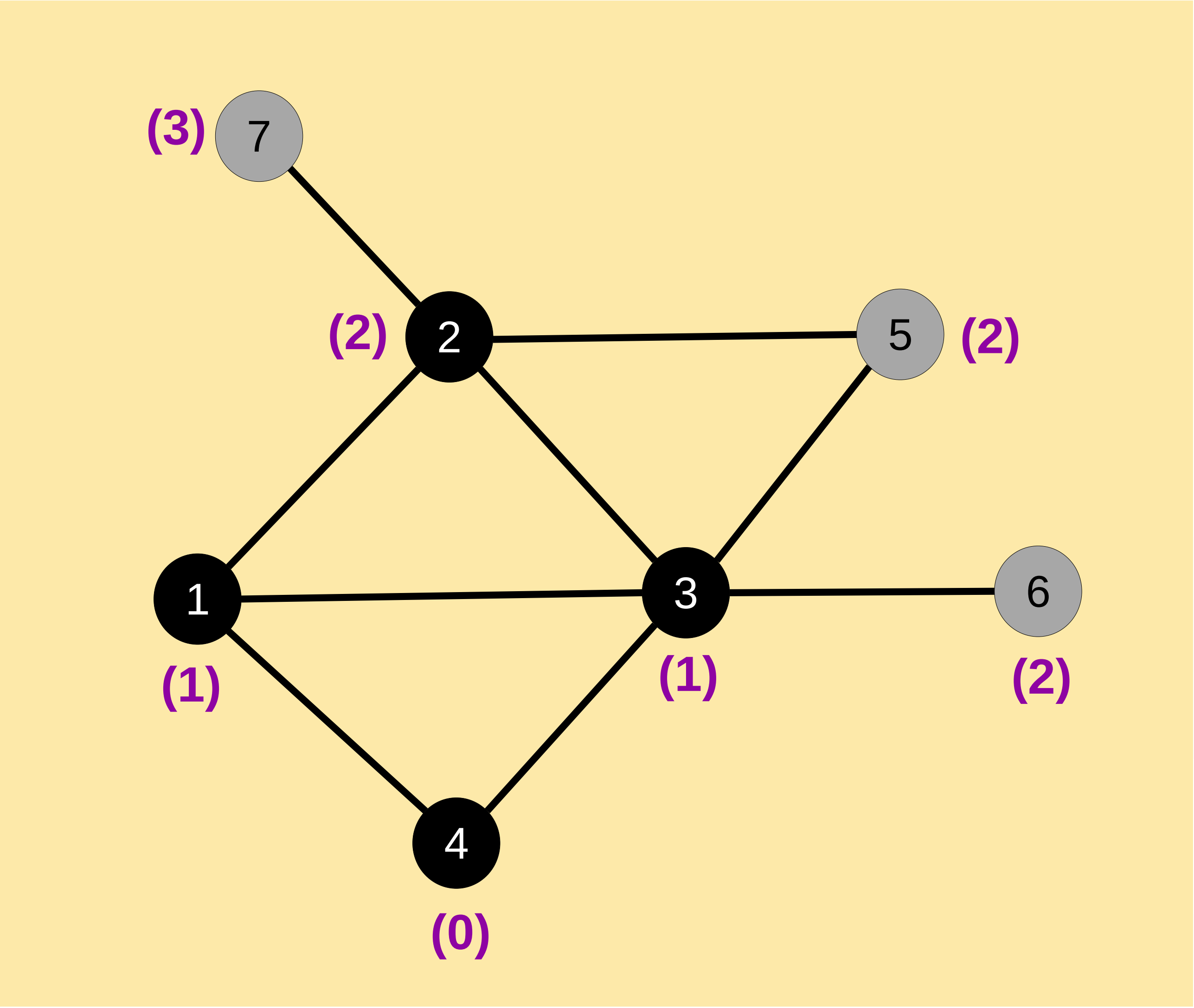


Fila = {6,5}

Predecessores:
1: 4
2: 1
3: 4
4: --
5: 3
6: 3
7:

Busca em largura (origem s=4)

Passo 8

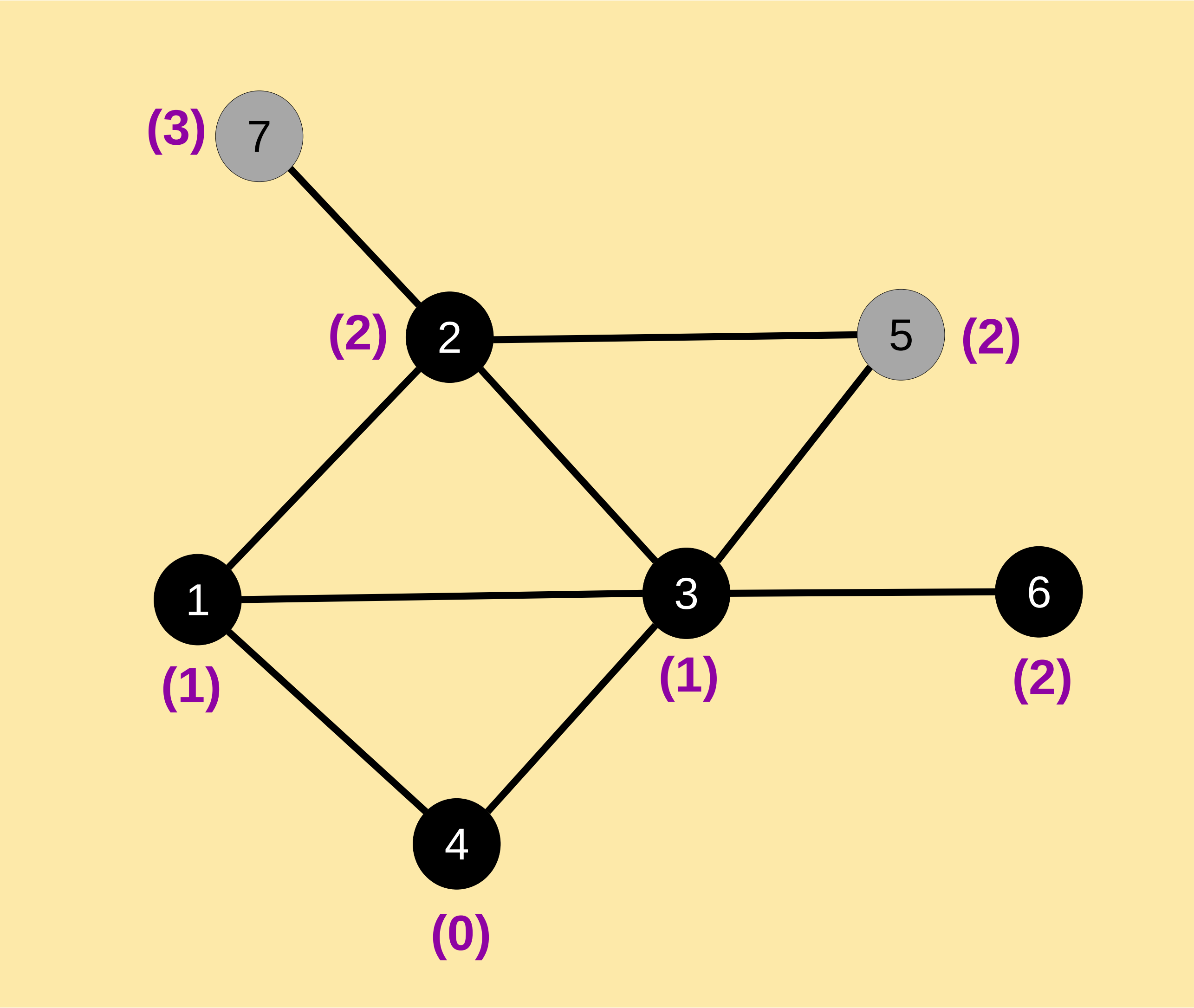


Fila = {6,5,7}

Predecessores:
1: 4
2: 1
3: 4
4: --
5: 3
6: 3
7: 2

Busca em largura (origem s=4)

Passo 9

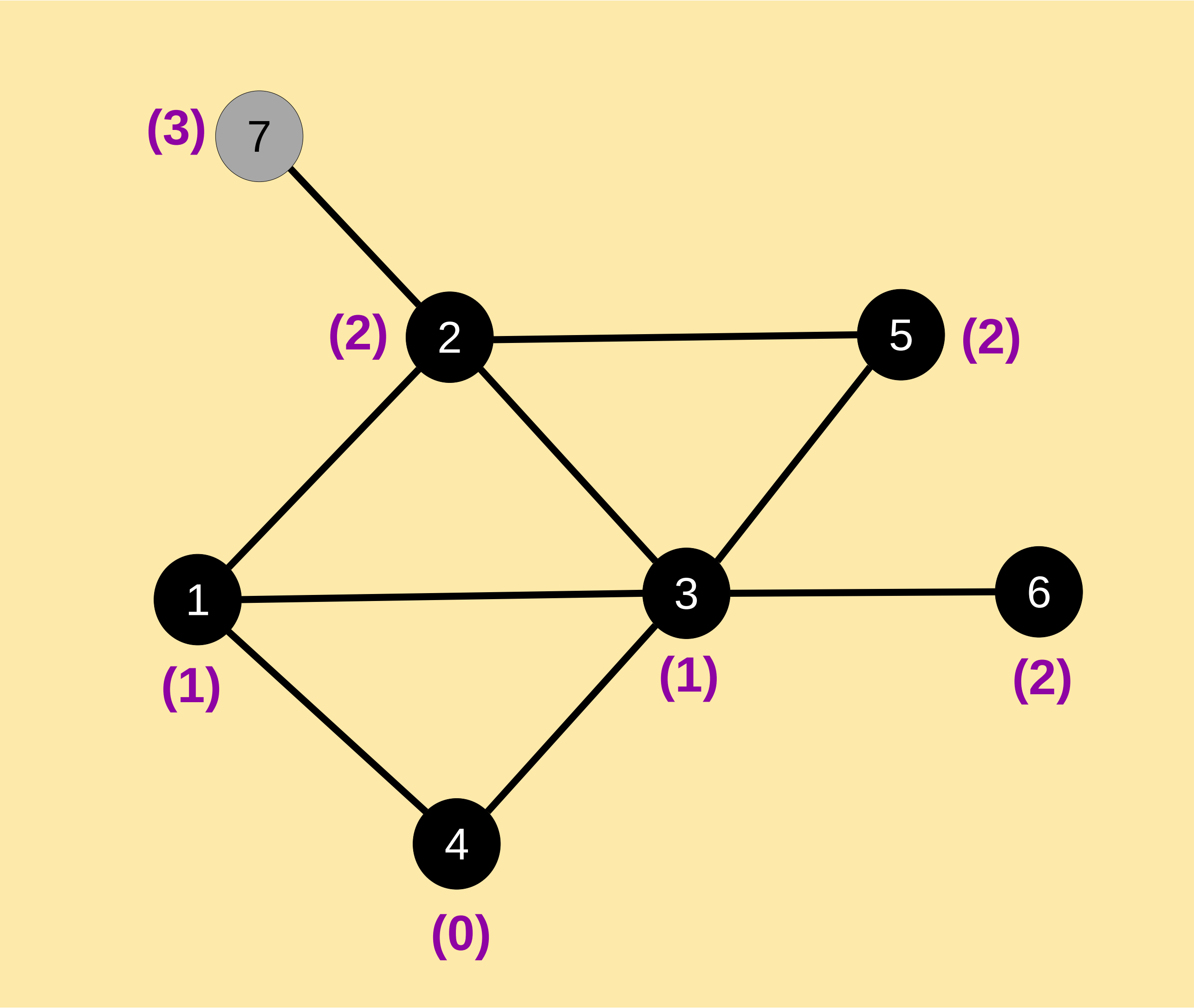


Fila = {5,7}

Predecessores:
1: 4
2: 1
3: 4
4: --
5: 3
6: 3
7: 2

Busca em largura (origem s=4)

Passo 10

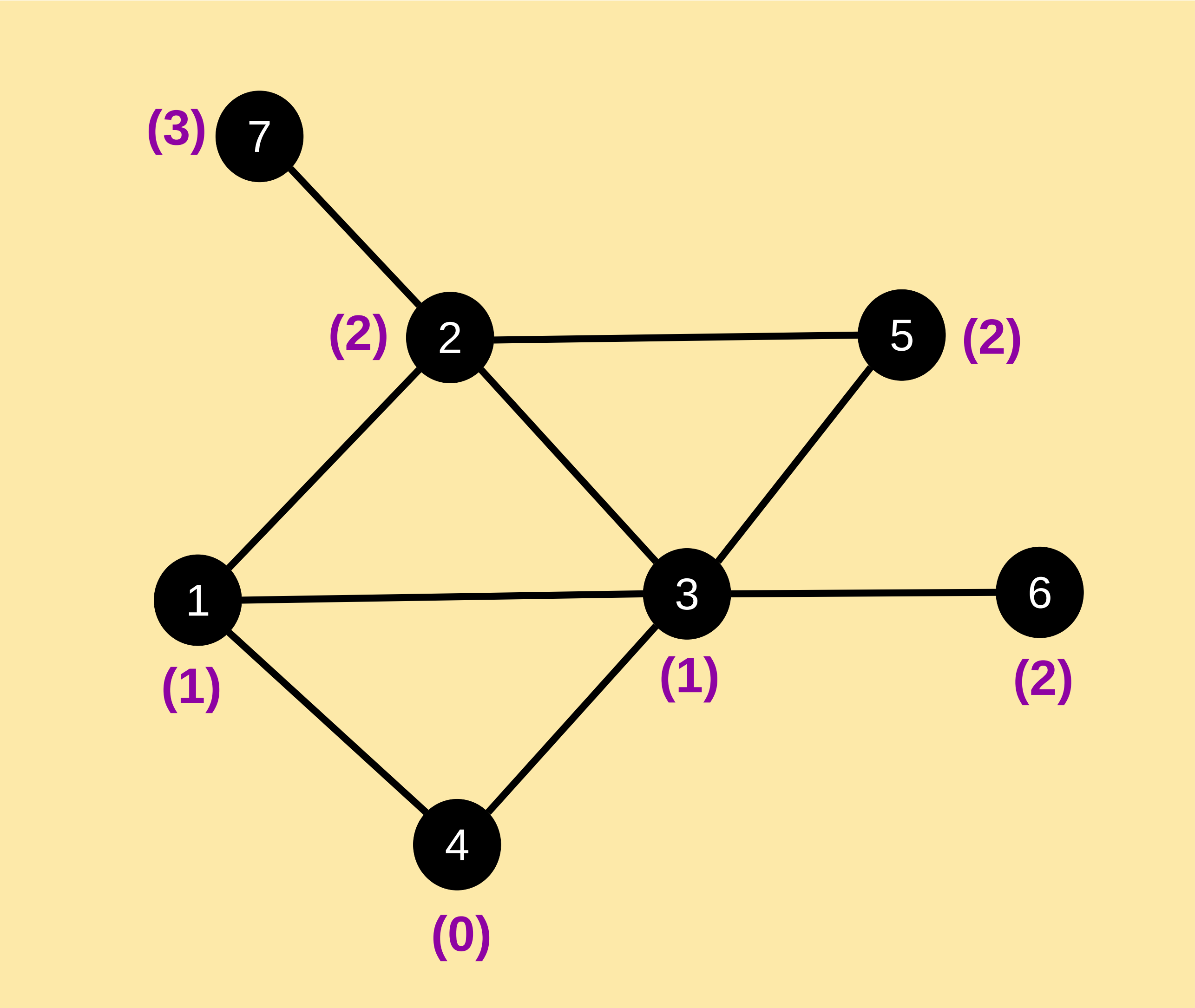


Fila = {7}

Predecessores:
1: 4
2: 1
3: 4
4: --
5: 3
6: 3
7: 2

Busca em largura (origem s=4)

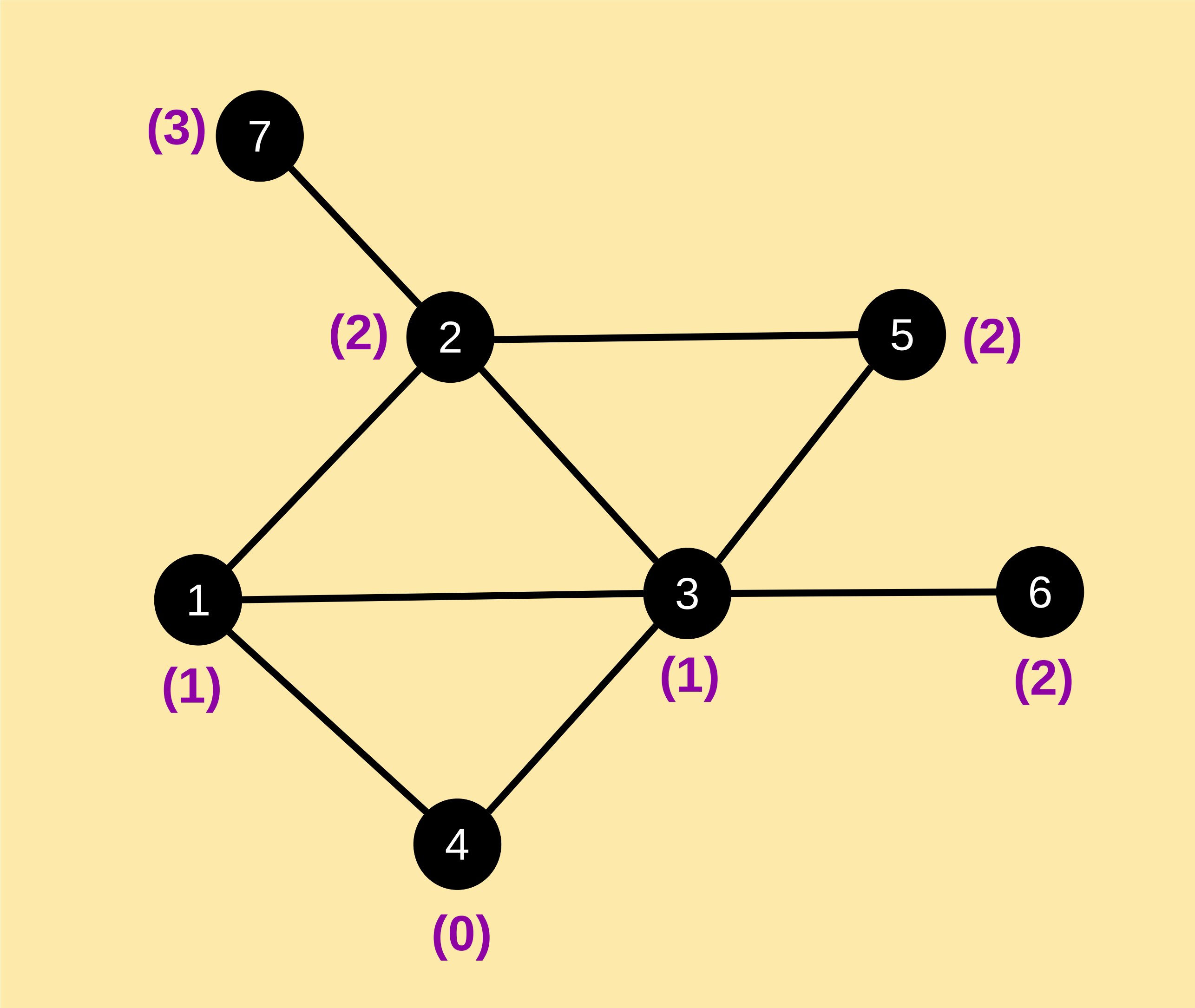
Passo 11



Fila = {}

Predecessores:
1: 4
2: 1
3: 4
4: --
5: 3
6: 3
7: 2

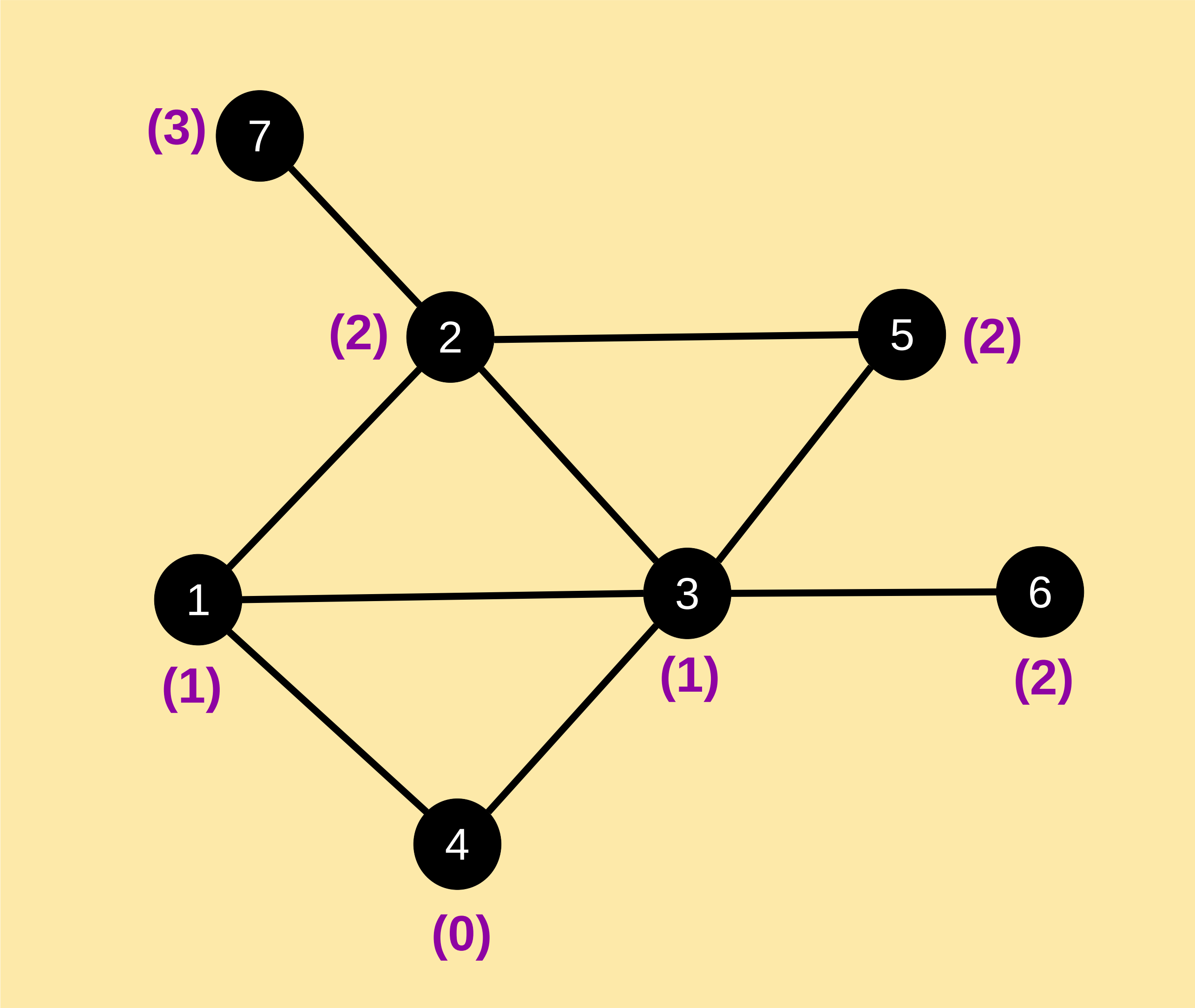
Busca em largura (origem s=4)



Predecessores:
1: 4
2: 1
3: 4
4: --
5: 3
6: 3
7: 2

Caminho exato para: a partir do vértice **4** chegar ao vértice **5**?

Busca em largura (origem s=4)



Predecessores:
1: 4
2: 1
3: 4
4: --
5: 3
6: 3
7: 2

Caminho exato para: a partir do vértice **4** chegar ao vértice **5**?

5 → 3 → 4

O caminho será <4,3,5>

Para o caminho entre a origem e o destino, parte-se do vértice de destino até chegar à origem. Depois inverte-se o vetor obtido.

Busca em largura (Algoritmo)

Constantes:

- BRANCO , CINZA ,PRETO, INFINITO

Variáveis:

- Q (fila) , s (vértice de origem)

Propriedades do vértice v:

- v.cor (cor do vértice)
- v.dis (distância do vértice **v** até a origem **s**)
- v.pre (precedessor do vértice **v**)

Funções :

- Insere(Q,v), permite inserir o vértice **v** na fila **Q**.
- Remove(Q), permite remover o primeiro vértice da fila **Q**.

Busca em largura (Algoritmo)

Busca em largura = Breadth First Search (BFS)

```
BFS(G,s):
  Para cada vértice v em G.V-{s} faça
      v.cor = BRANCO
      v.dis = INFINITO
  s.cor = CINZA
  s.dis = 0
  Q = VAZIO
  Insere(Q,s)

  Enquanto Q ≠ VAZIO faça
      u = Remove(Q)
      Para cada vértice v em G.Adj[u] faça
          se v.cor == BRANCO
              v.cor = CINZA
              v.dis = u.dis+1
              v.pre = u
              Insere(Q,v)
      u.cor = PRETO
```

Busca em largura (Algoritmo)

Busca em largura = Breadth First Search (BFS)

Inicialização

```
BFS(G,s):
   Para cada vértice v em G.V-{s} faça
       v.cor = BRANCO
       v.dis = INFINITO
   s.cor = CINZA
   s.dis = 0
   Q = VAZIO
   Insere(Q,s)
```

Percorre o grafo

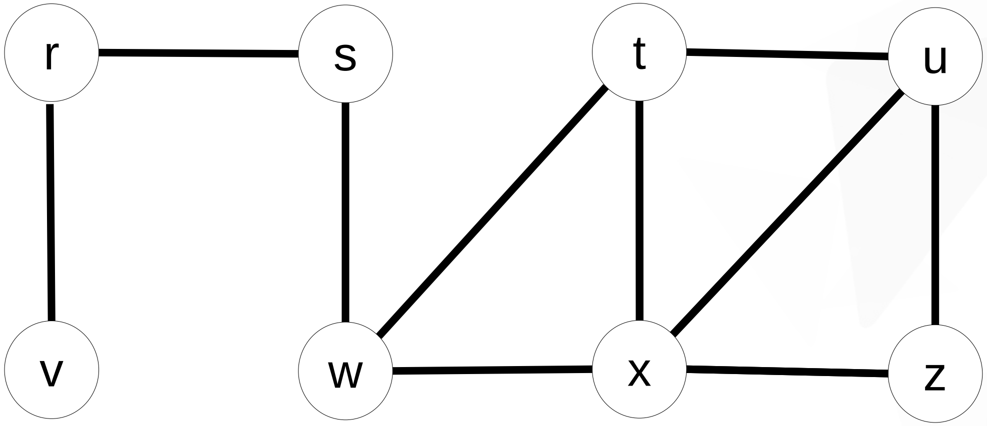
Explora os vizinhos de u

```
   Enquanto Q ≠ VAZIO faça
       u = Remove(Q)
       Para cada vértice v em G.Adj[u] faça
           se v.cor == BRANCO
               v.cor = CINZA
               v.dis = u.dis+1
               v.pre = u
               Insere(Q,v)
       u.cor = PRETO
```

Busca em largura (Algoritmo)

```
Para cada vértice v em G.V-{s} faça
    v.cor = BRANCO
    v.dis = INFINITO
s.cor = CINZA
s.dis = 0
Q = VAZIO
Insere(Q,s)
```

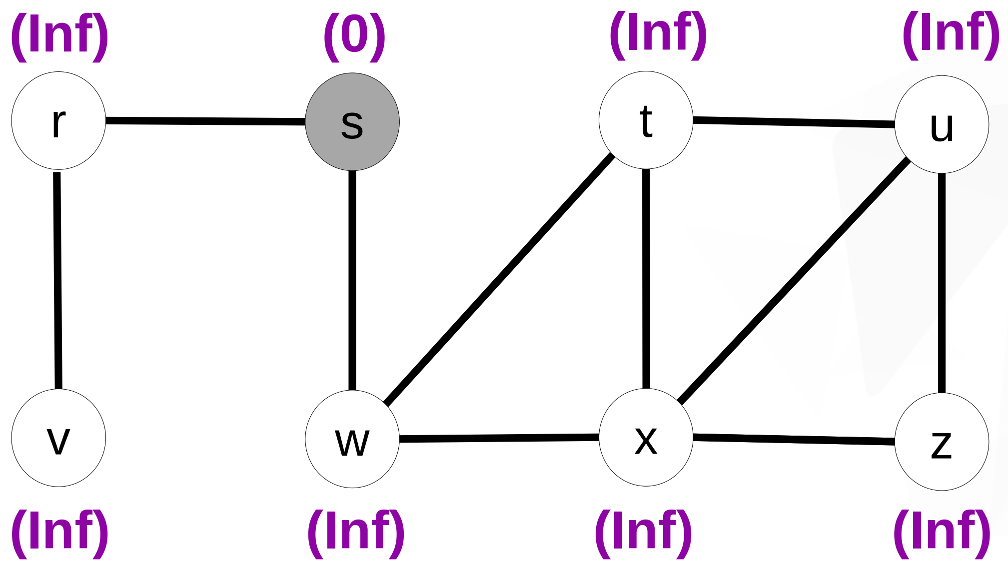
Inicialização



Busca em largura (Algoritmo)

Inicialização

```
Para cada vértice v em G.V-{s} faça
    v.cor = BRANCO
    v.dis = INFINITO
s.cor = CINZA
s.dis = 0
Q = VAZIO
Insere(Q,s)
```



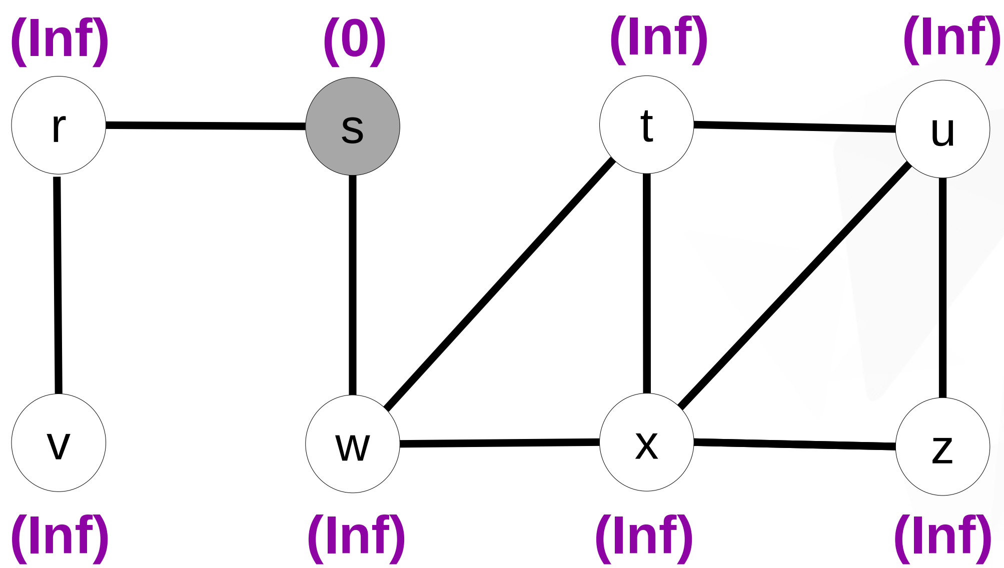
Q = {s}

Busca em largura (Algoritmo)

```
Enquanto Q ≠ VAZIO faça
    u = Remove(Q)
    Para cada vértice v em G.Adj[u] faça
        se v.cor == BRANCO
            v.cor = CINZA
            v.dis = u.dis+1
            v.pre = u
            Insere(Q,v)
    u.cor = PRETO
```

Percorre o grafo

Explora os vizinhos de u



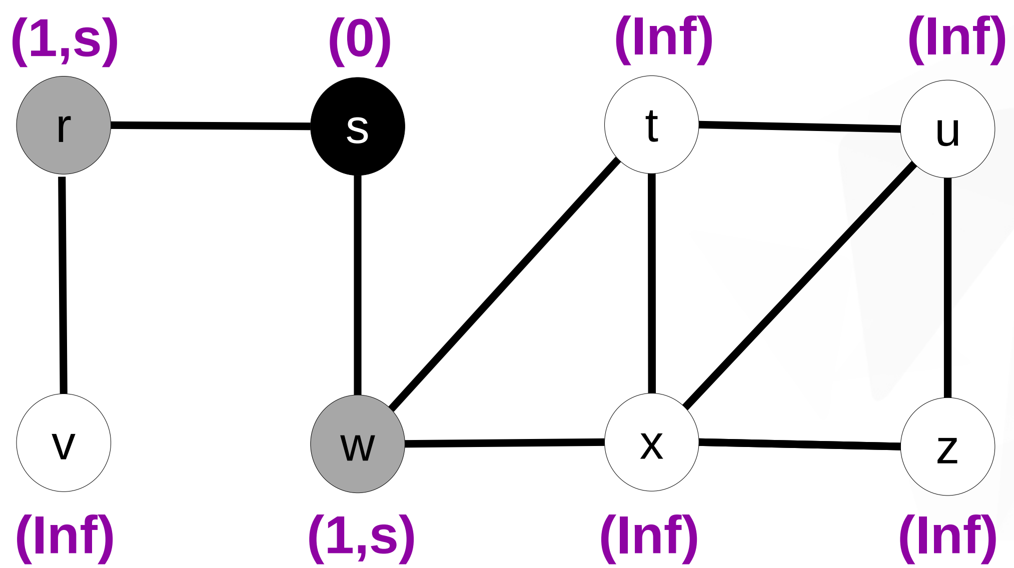
Q = {s}

Busca em largura (Algoritmo)

```
Enquanto Q ≠ VAZIO faça
    u = Remove(Q)
    Para cada vértice v em G.Adj[u] faça
        se v.cor == BRANCO
            v.cor = CINZA
            v.dis = u.dis+1
            v.pre = u
            Insere(Q,v)
    u.cor = PRETO
```

Percorre o grafo

Explora os vizinhos de u



Q = {w,r}

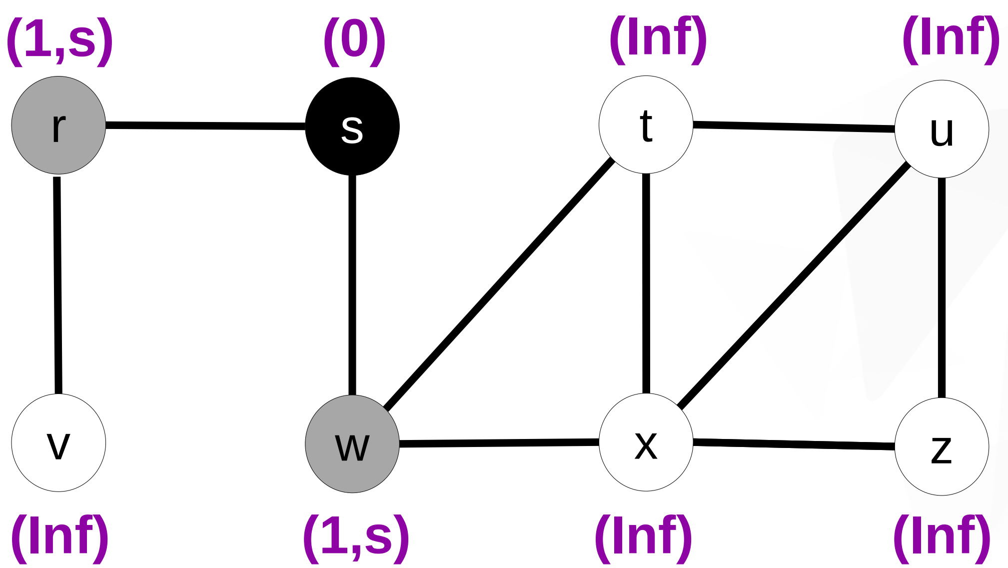
u = s

Busca em largura (Algoritmo)

```
Enquanto Q ≠  VAZIO faça
    u = Remove(Q)
    Para cada vértice v em G.Adj[u] faça
        se v.cor == BRANCO
            v.cor = CINZA
            v.dis = u.dis+1
            v.pre = u
            Insere(Q,v)
    u.cor = PRETO
```

Percorre o grafo

Explora os vizinhos de u



Q = {r}

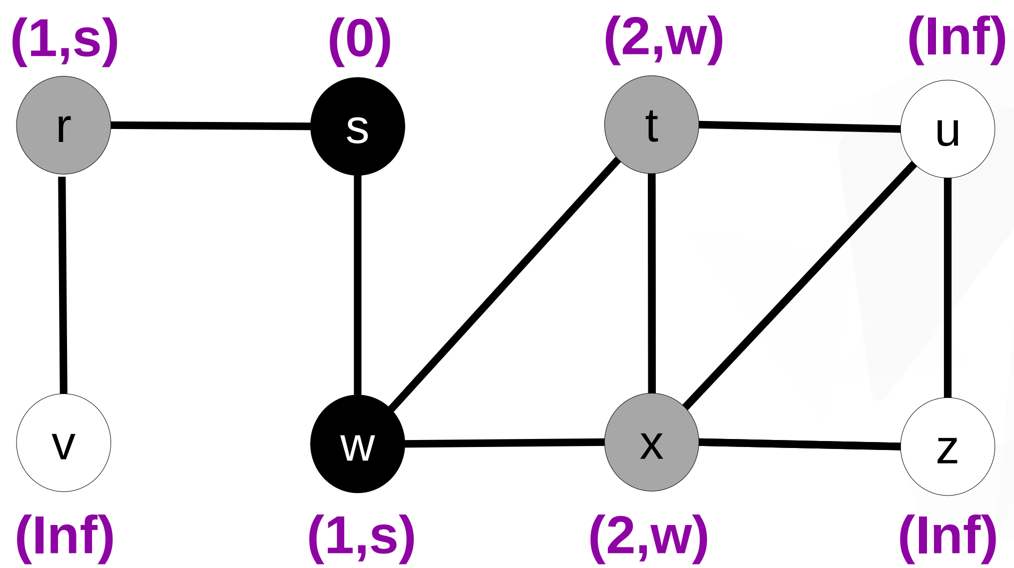
u = w

Busca em largura (Algoritmo)

```
Enquanto Q ≠ VAZIO faça
    u = Remove(Q)
    Para cada vértice v em G.Adj[u] faça
        se v.cor == BRANCO
            v.cor = CINZA
            v.dis = u.dis+1
            v.pre = u
            Insere(Q,v)
    u.cor = PRETO
```

Percorre o grafo

Explora os vizinhos de u



Q = {r,t,x}

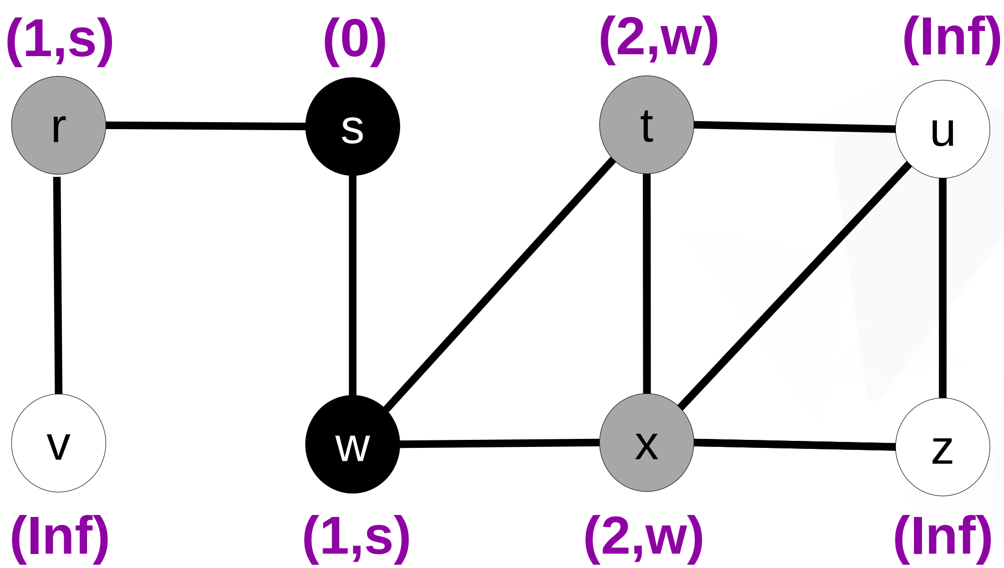
u = w

Busca em largura (Algoritmo)

```
Enquanto Q ≠ VAZIO faça
    u = Remove(Q)
    Para cada vértice v em G.Adj[u] faça
        se v.cor == BRANCO
            v.cor = CINZA
            v.dis = u.dis+1
            v.pre = u
            Insere(Q,v)
    u.cor = PRETO
```

Percorre o grafo

Explora os vizinhos de u



Q = {t,x}

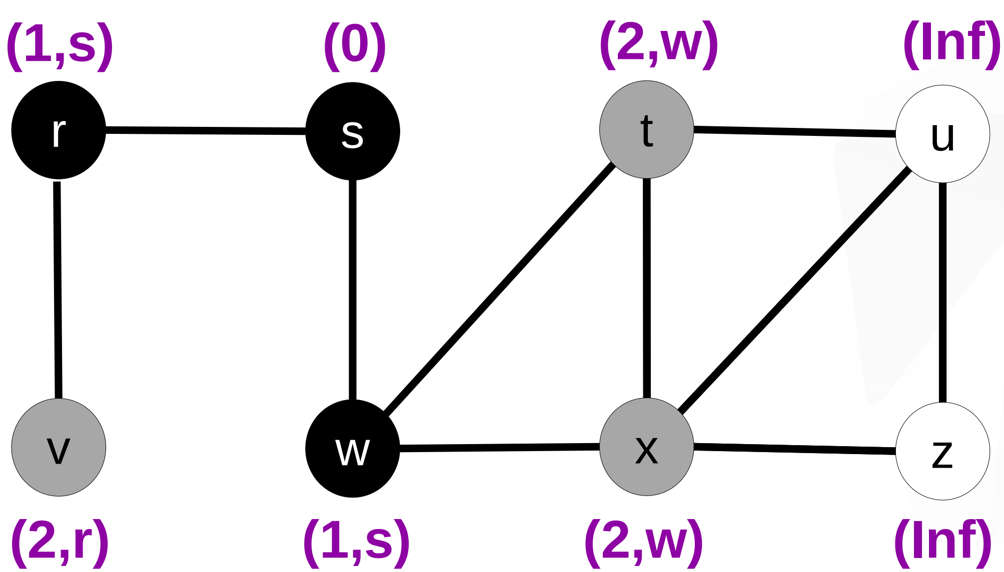
u = r

Busca em largura (Algoritmo)

Percorre o grafo

```
Enquanto Q ≠ VAZIO faça
    u = Remove(Q)
    Para cada vértice v em G.Adj[u] faça
        se v.cor == BRANCO
            v.cor = CINZA
            v.dis = u.dis+1
            v.pre = u
            Insere(Q,v)
    u.cor = PRETO
```

Explora os vizinhos de u



Q = {t,x,v}

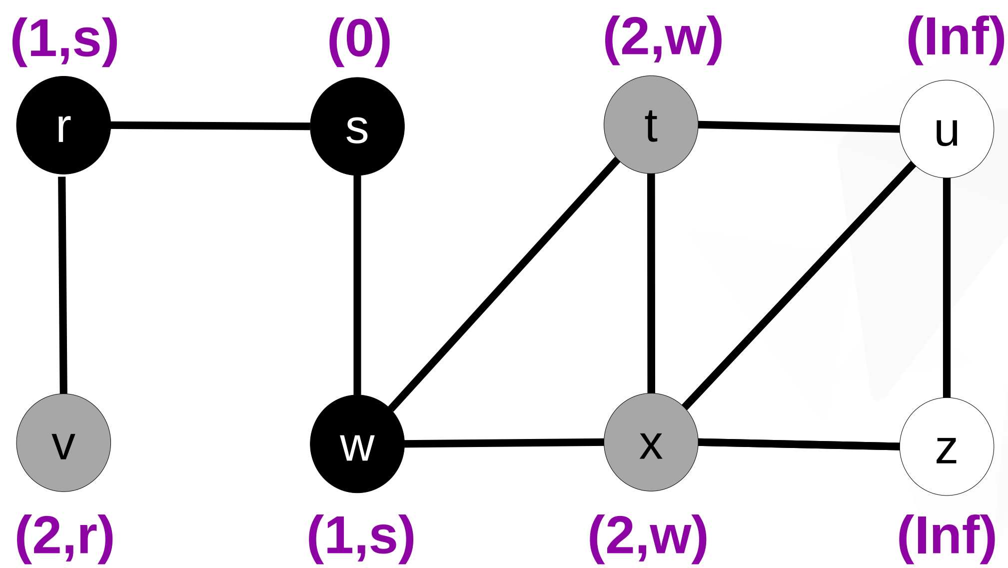
u = r

Busca em largura (Algoritmo)

Percorre o grafo

```
Enquanto Q ≠ VAZIO faça
    u = Remove(Q)
    Para cada vértice v em G.Adj[u] faça
        se v.cor == BRANCO
            v.cor = CINZA
            v.dis = u.dis+1
            v.pre = u
            Insere(Q,v)
    u.cor = PRETO
```

Explora os vizinhos de u



Q = {x,v}

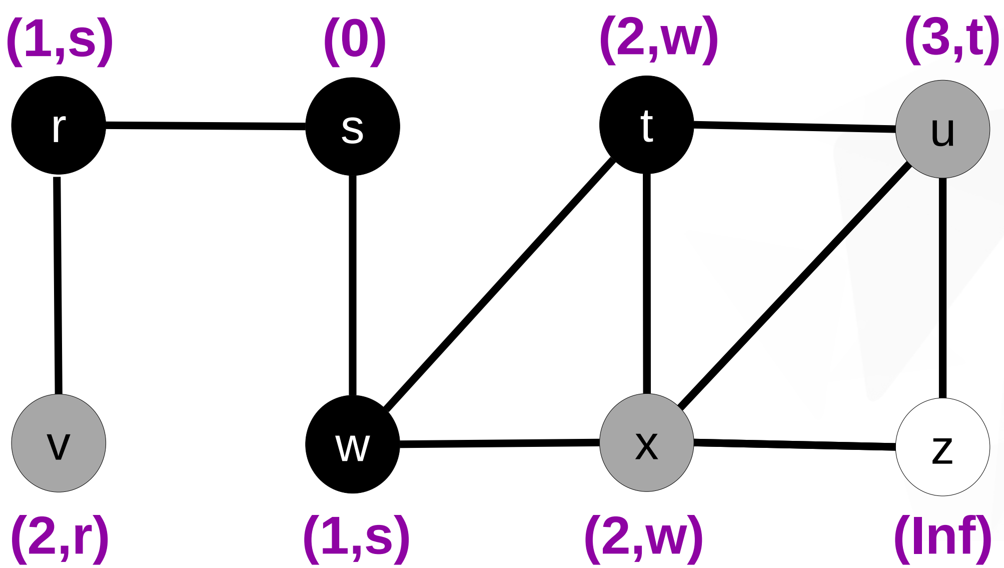
u = t

Busca em largura (Algoritmo)

Percorre o grafo

```
Enquanto Q ≠ VAZIO faça
    u = Remove(Q)
    Para cada vértice v em G.Adj[u] faça
        se v.cor == BRANCO
            v.cor = CINZA
            v.dis = u.dis+1
            v.pre = u
            Insere(Q,v)
    u.cor = PRETO
```

Explora os vizinhos de u



Q = {x,v,u}

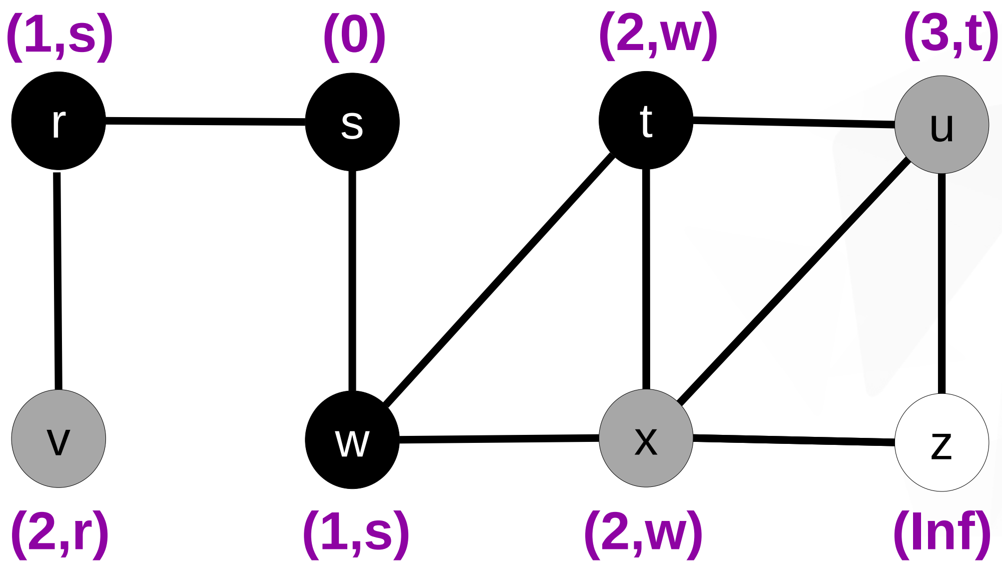
u = t

Busca em largura (Algoritmo)

Percorre o grafo

```
Enquanto Q ≠  VAZIO faça
    u = Remove(Q)
    Para cada vértice v em G.Adj[u] faça
        se v.cor == BRANCO
            v.cor = CINZA
            v.dis = u.dis+1
            v.pre = u
            Insere(Q,v)
    u.cor = PRETO
```

Explora os vizinhos de u



Q = {v,u}

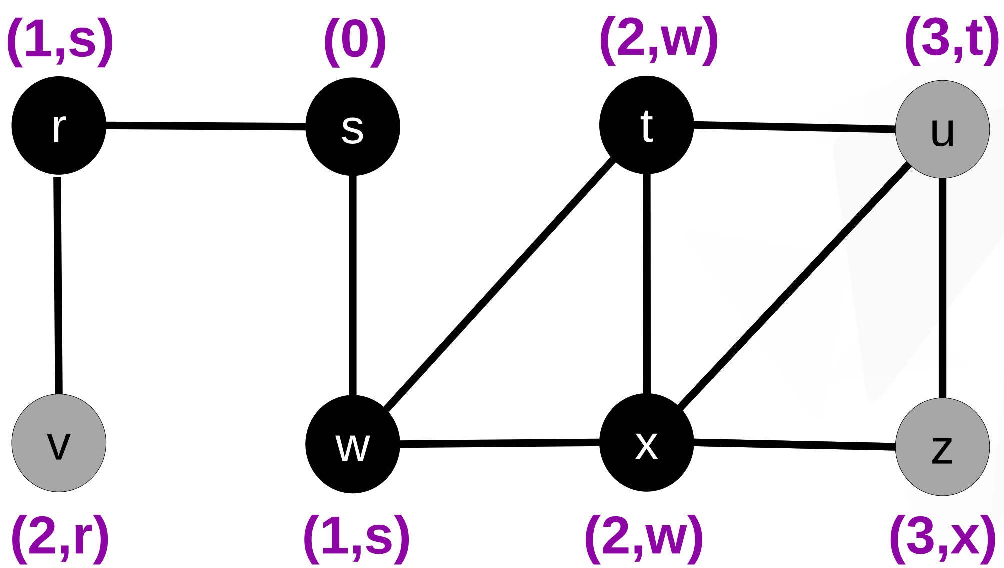
u = x

Busca em largura (Algoritmo)

```
Enquanto Q ≠ VAZIO faça
    u = Remove(Q)
    Para cada vértice v em G.Adj[u] faça
        se v.cor == BRANCO
            v.cor = CINZA
            v.dis = u.dis+1
            v.pre = u
            Insere(Q,v)
    u.cor = PRETO
```

Percorre o grafo

Explora os vizinhos de u



Q = {v,u,z}

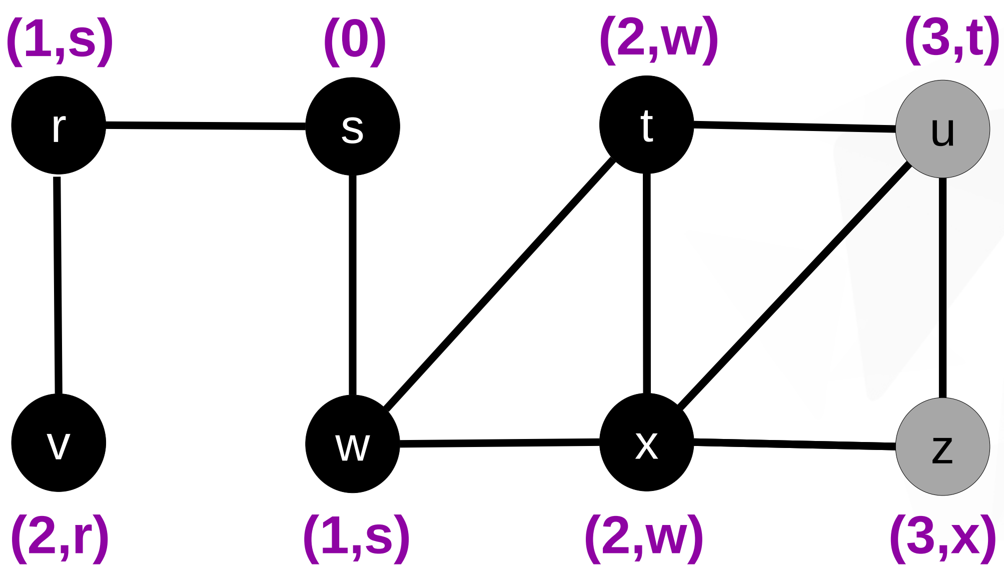
u = x

Busca em largura (Algoritmo)

Percorre o grafo

```
Enquanto Q ≠ VAZIO faça
    u = Remove(Q)
    Para cada vértice v em G.Adj[u] faça
        se v.cor == BRANCO
            v.cor = CINZA
            v.dis = u.dis+1
            v.pre = u
            Insere(Q,v)
    u.cor = PRETO
```

Explora os vizinhos de u



Q = {u,z}

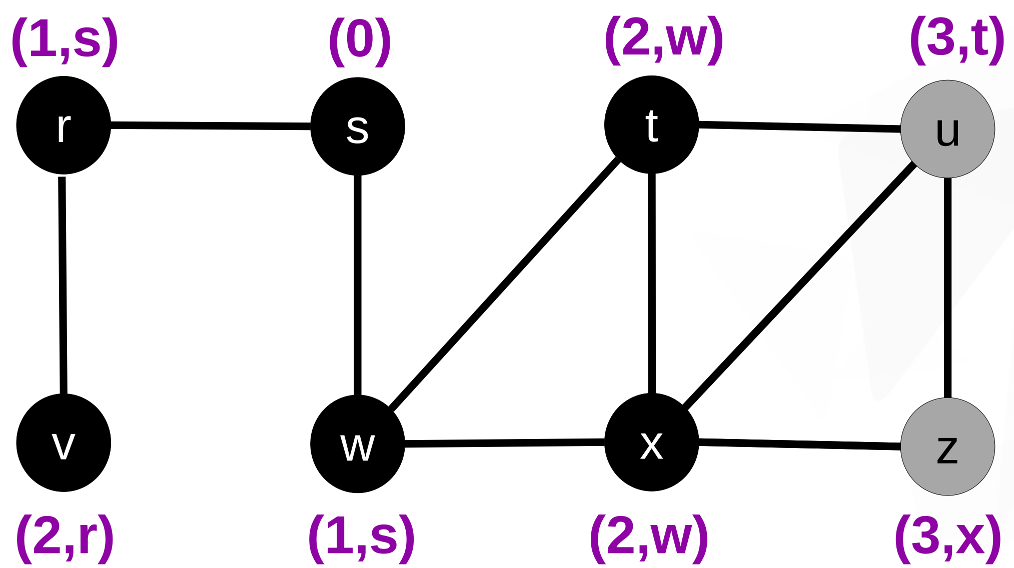
u = v

Busca em largura (Algoritmo)

```
Enquanto Q ≠ VAZIO faça
    u = Remove(Q)
    Para cada vértice v em G.Adj[u] faça
        se v.cor == BRANCO
            v.cor = CINZA
            v.dis = u.dis+1
            v.pre = u
            Insere(Q,v)
    u.cor = PRETO
```

Percorre o grafo

Explora os vizinhos de u



Q = {z}

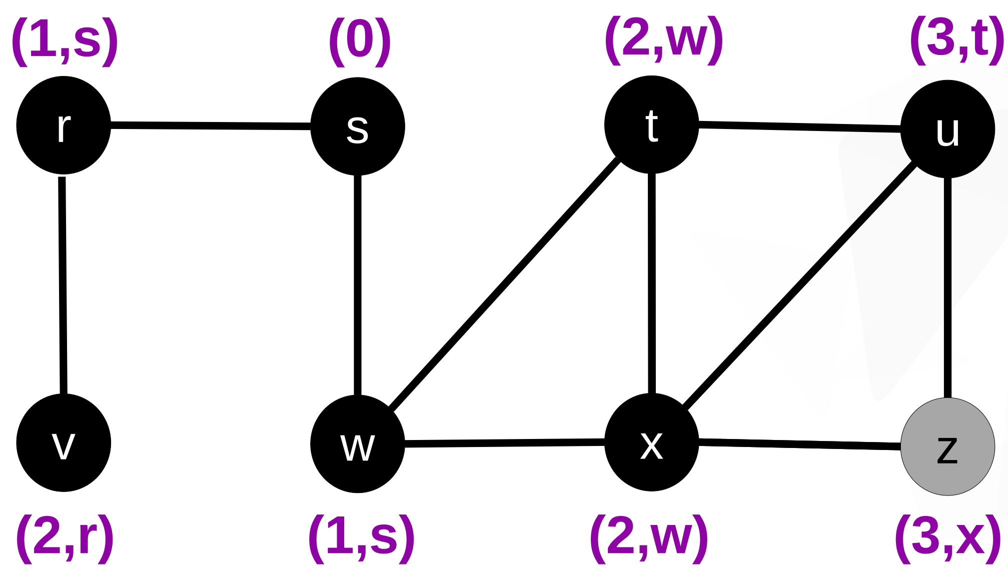
u = u

Busca em largura (Algoritmo)

Percorre o grafo

```
Enquanto Q ≠ VAZIO faça
    u = Remove(Q)
    Para cada vértice v em G.Adj[u] faça
        se v.cor == BRANCO
            v.cor = CINZA
            v.dis = u.dis+1
            v.pre = u
            Insere(Q,v)
    u.cor = PRETO
```

Explora os vizinhos de u



Q = {z}

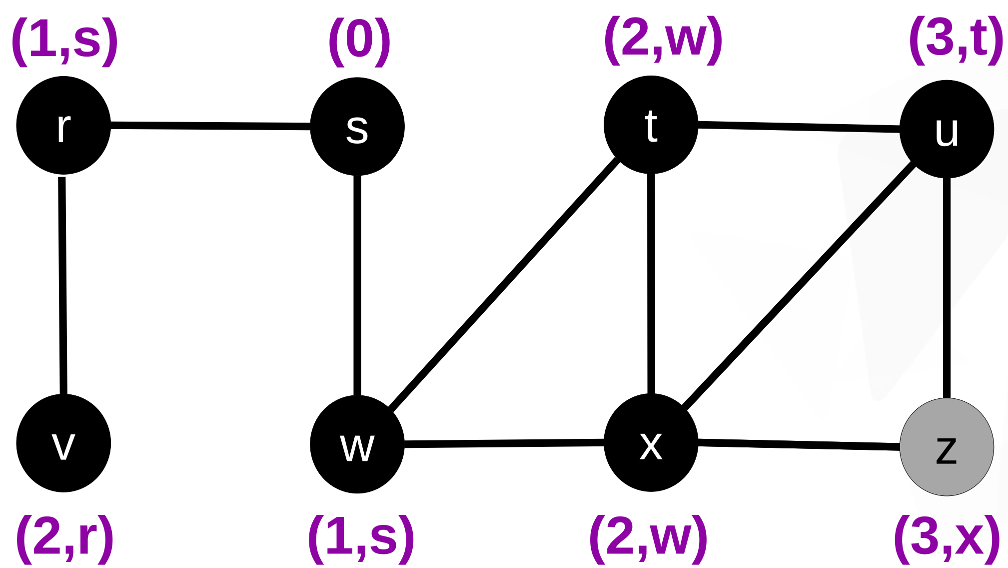
u = u

Busca em largura (Algoritmo)

```
Enquanto Q ≠  VAZIO faça
    u = Remove(Q)
    Para cada vértice v em G.Adj[u] faça
        se v.cor == BRANCO
            v.cor = CINZA
            v.dis = u.dis+1
            v.pre = u
            Insere(Q,v)
    u.cor = PRETO
```

Percorre o grafo

Explora os vizinhos de u



Q = {}

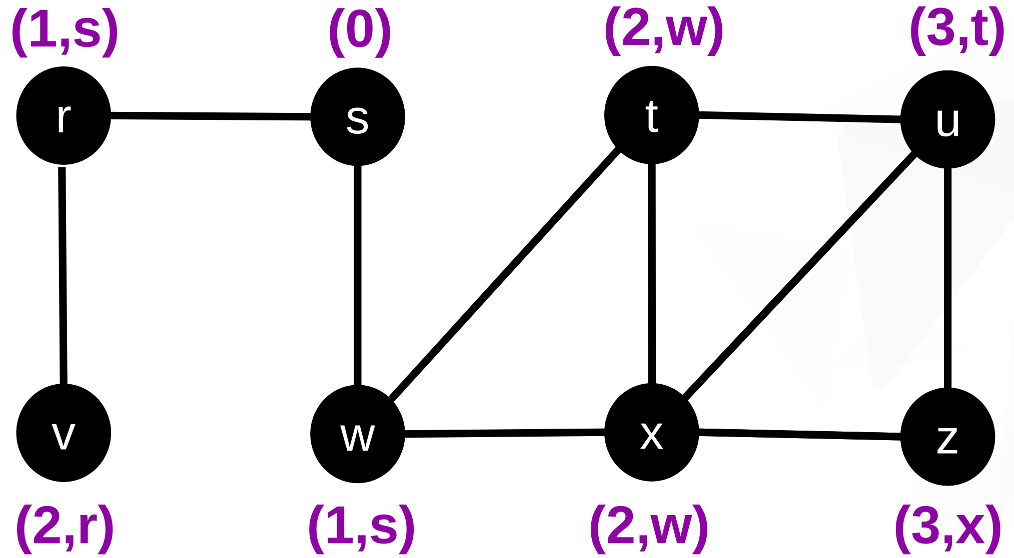
u = z

Busca em largura (Algoritmo)

Percorre o grafo

Explora os vizinhos de u

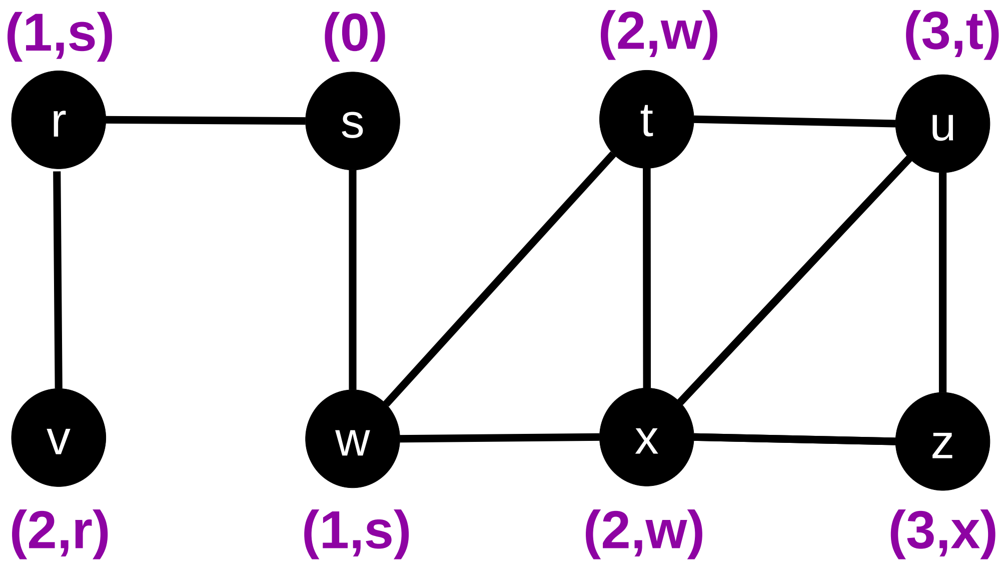
```
Enquanto Q ≠ VAZIO faça
    u = Remove(Q)
    Para cada vértice v em G.Adj[u] faça
        se v.cor == BRANCO
            v.cor = CINZA
            v.dis = u.dis+1
            v.pre = u
            Insere(Q,v)
    u.cor = PRETO
```



Q = {}

u = z

Busca em largura



Caminho mínimo do vértice **s** ao vértice **u:**

u.pre = t
t.pre = w
w.pre = s

Assim : <s,w,t,u>

Busca em largura

- O algoritmo de busca em grafos também **trabalha com grafos não conexos**.

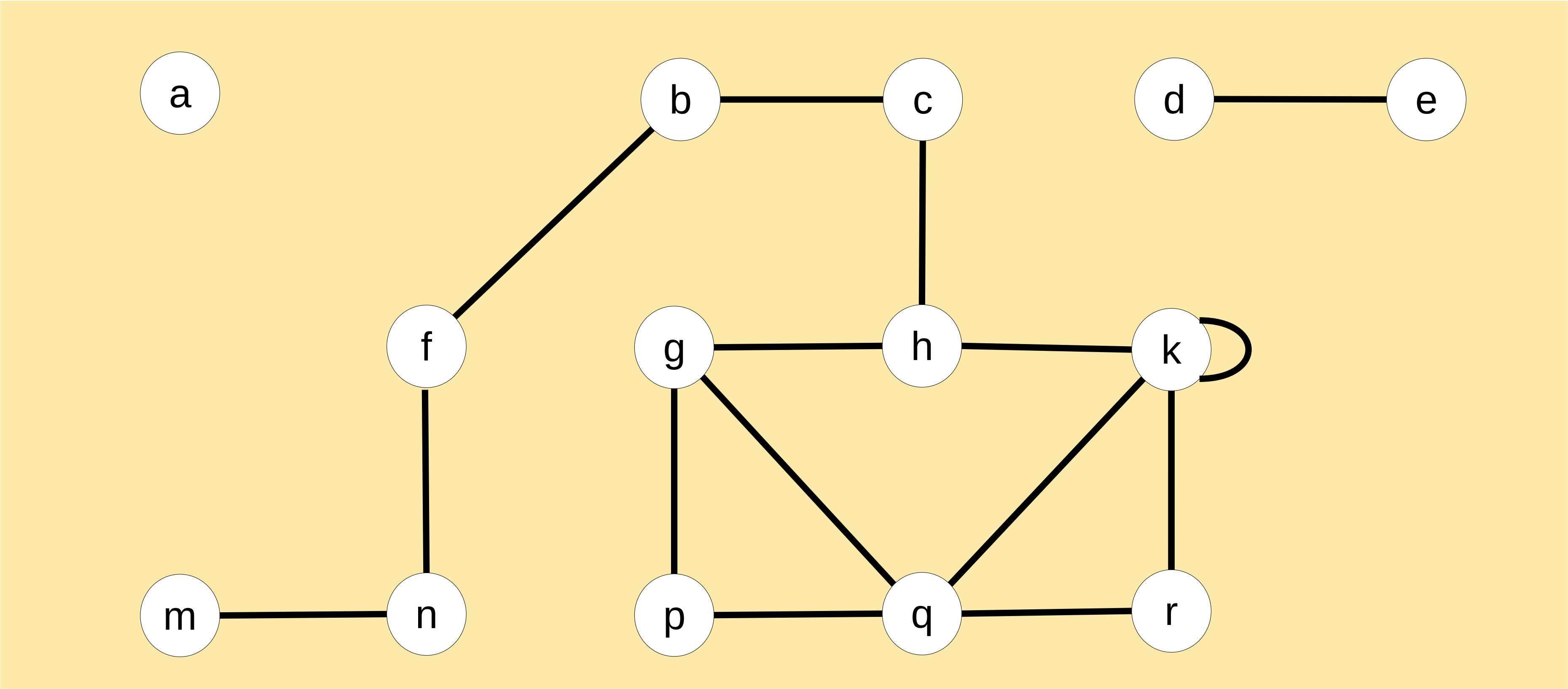
- Para tais casos, apenas os vértices **v** que **estão na mesma componente** do vértice **s** terão valores v.dis $\neq$ INFINITO.

- Podemos usar o atributo v.dis de todos os vértices para **decidir** se o grafo é conexo.

III. Atividade Prática

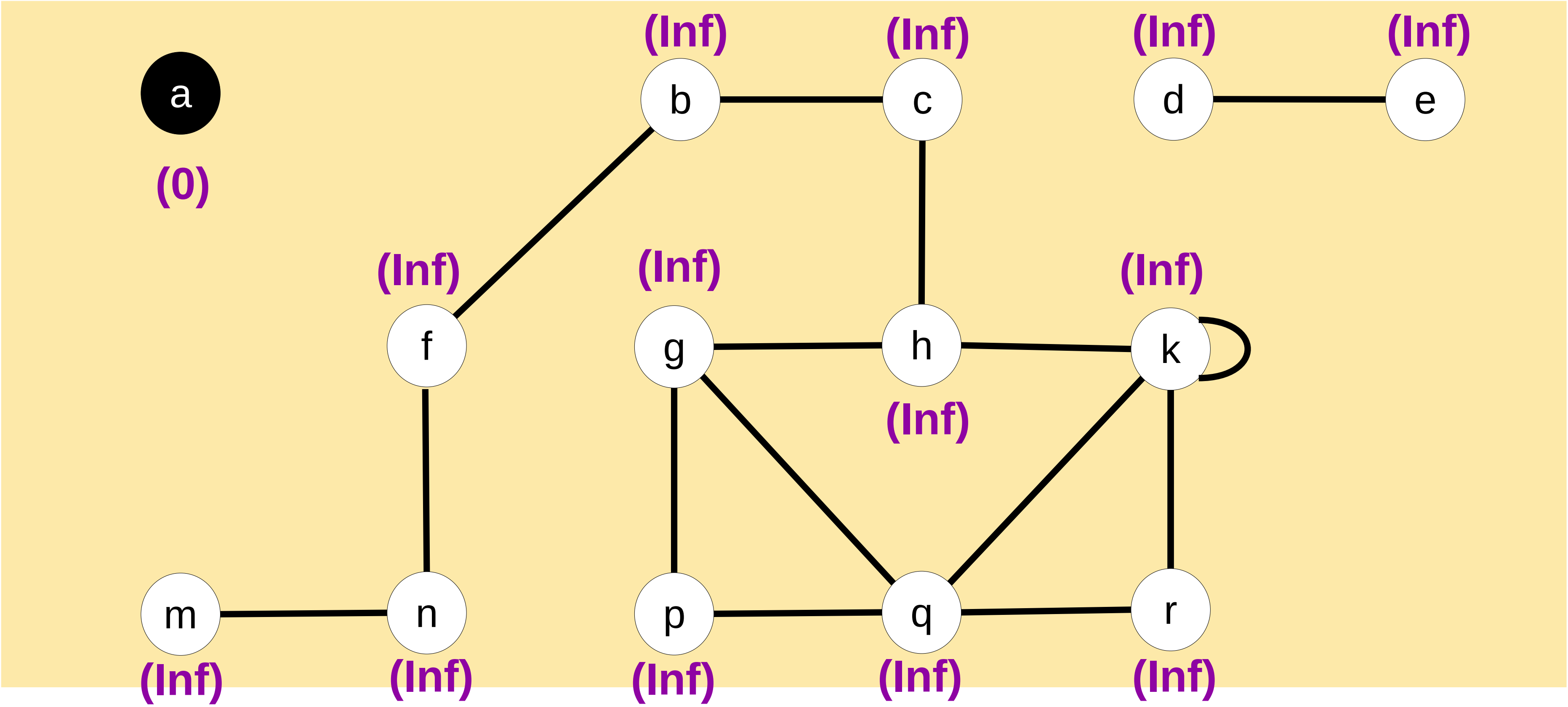
Atividade Prática

- **1.** Utilize o algoritmo de busca em largura para obter a distância:
  - **a)** Do vértice **a** ao vértice **q.**
  - **b)** Do vértice **c** ao vértice **q.**



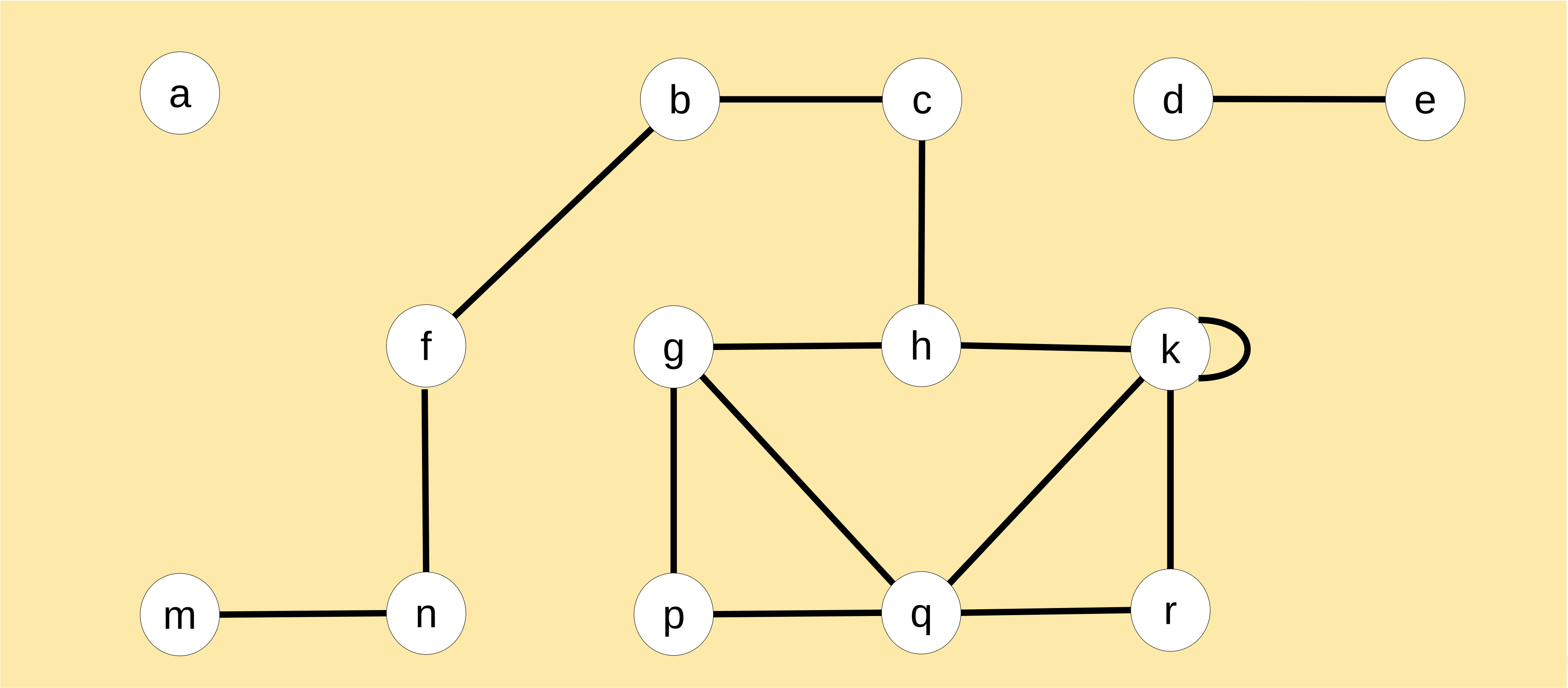
Atividade Prática

- **1.** Utilize o algoritmo de busca em largura para obter a distância:
  - **a)** Do vértice **a** ao vértice **q.**



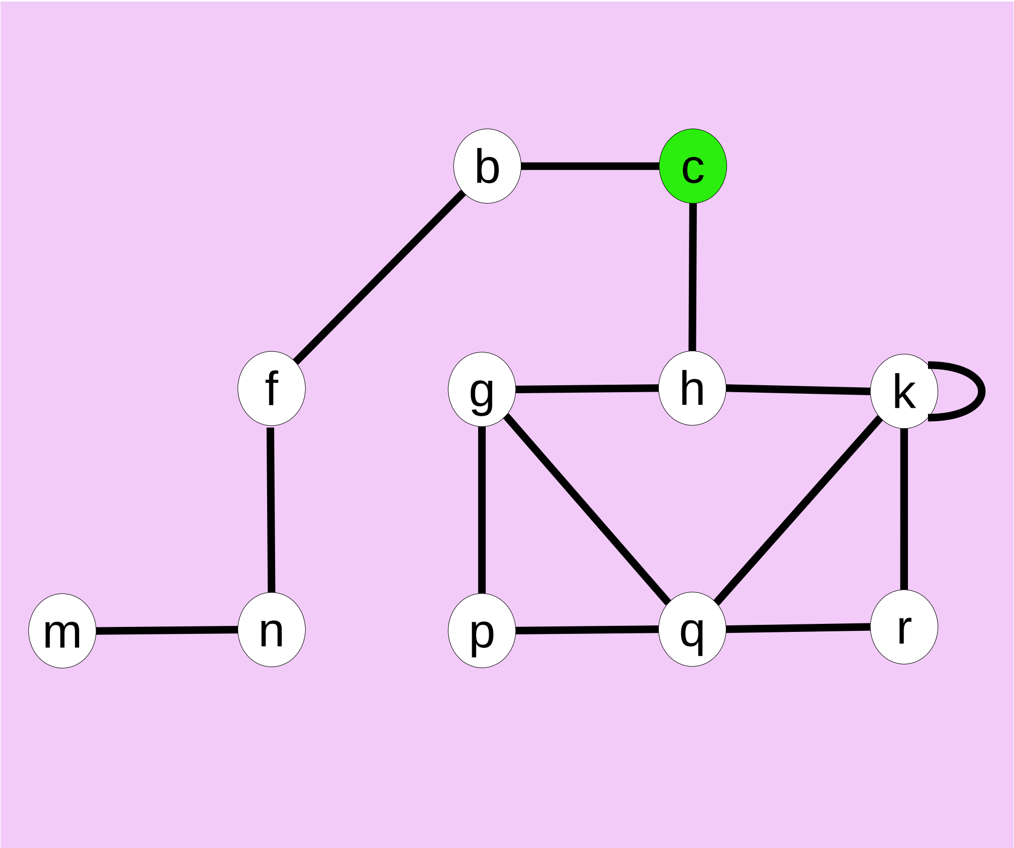
Atividade Prática

- **1.** Utilize o algoritmo de busca em largura para obter a distância:
  - **b)** Do vértice **c** ao vértice **q.**



Atividade prática

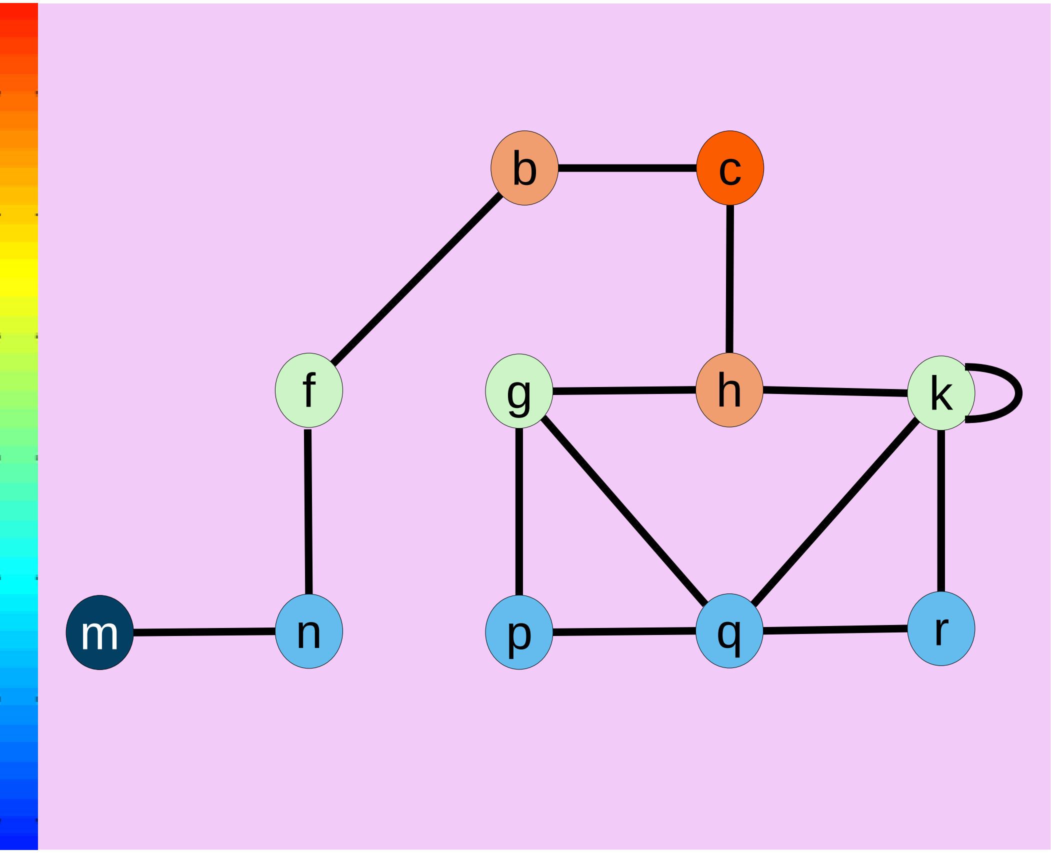
- Algoritmo de busca em largura iniciando do vértice c



Grafo de entrada

Atividade prática

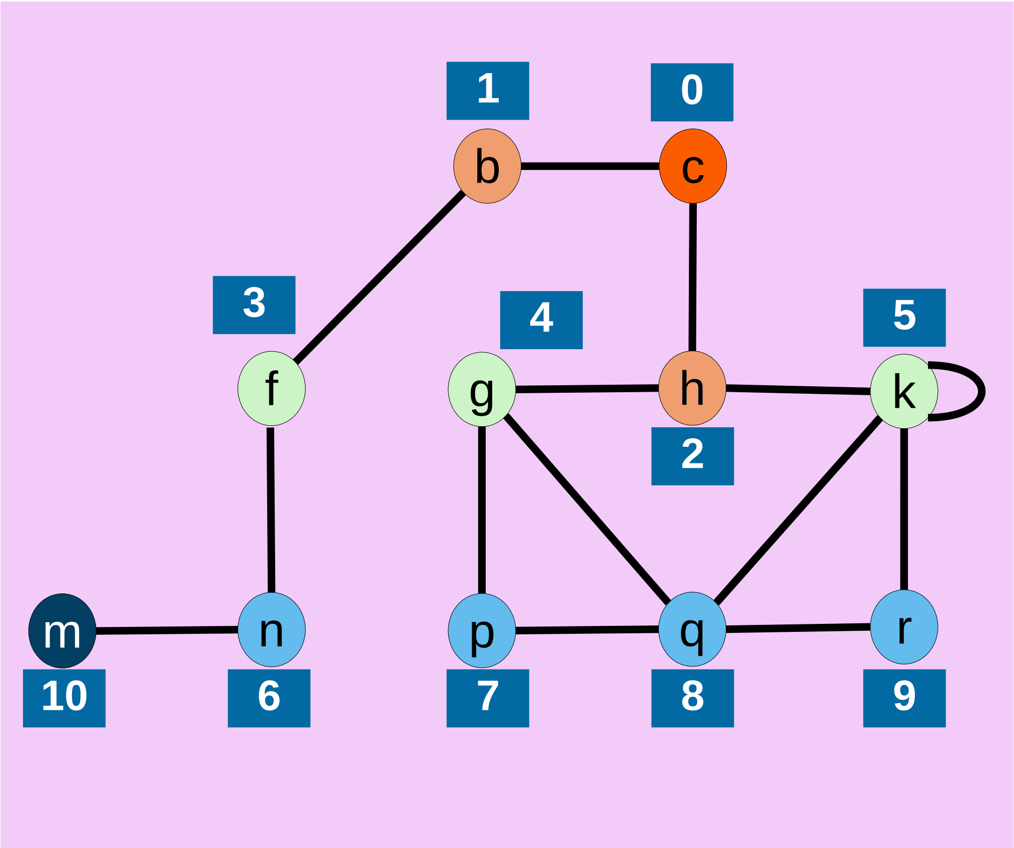
- Algoritmo de busca em largura iniciando do vértice c



Grafo de entrada

Atividade prática

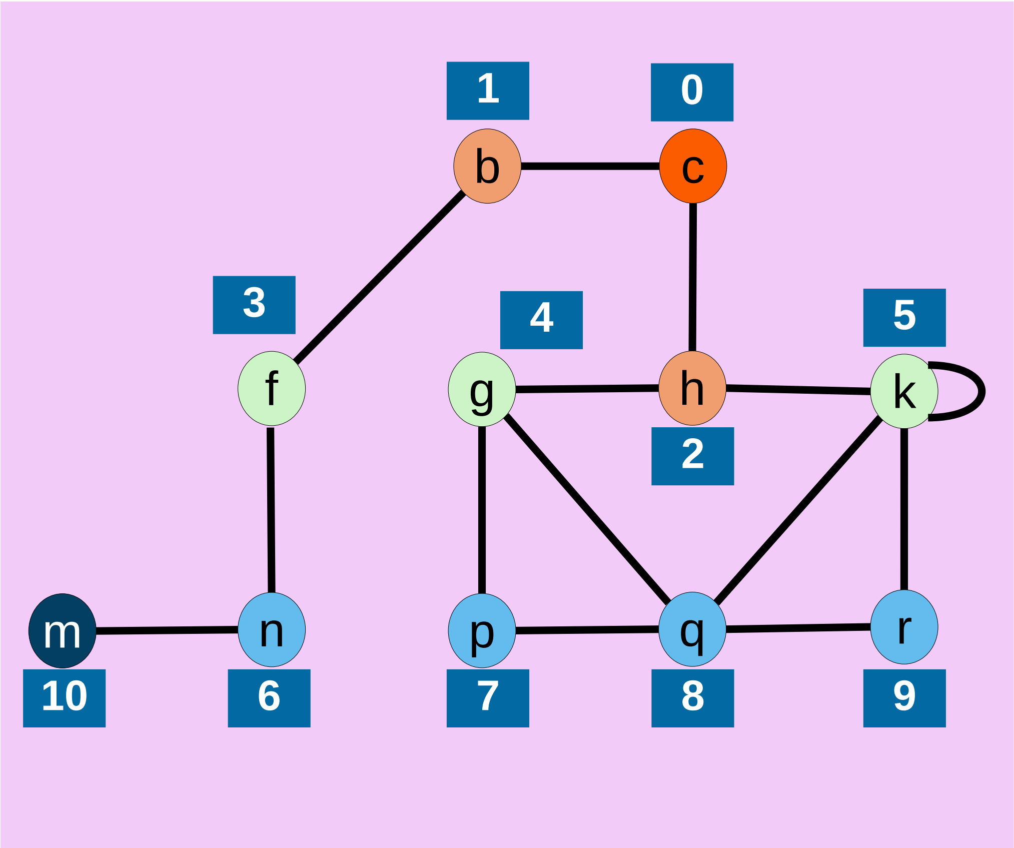
- Algoritmo de busca em largura iniciando do vértice c



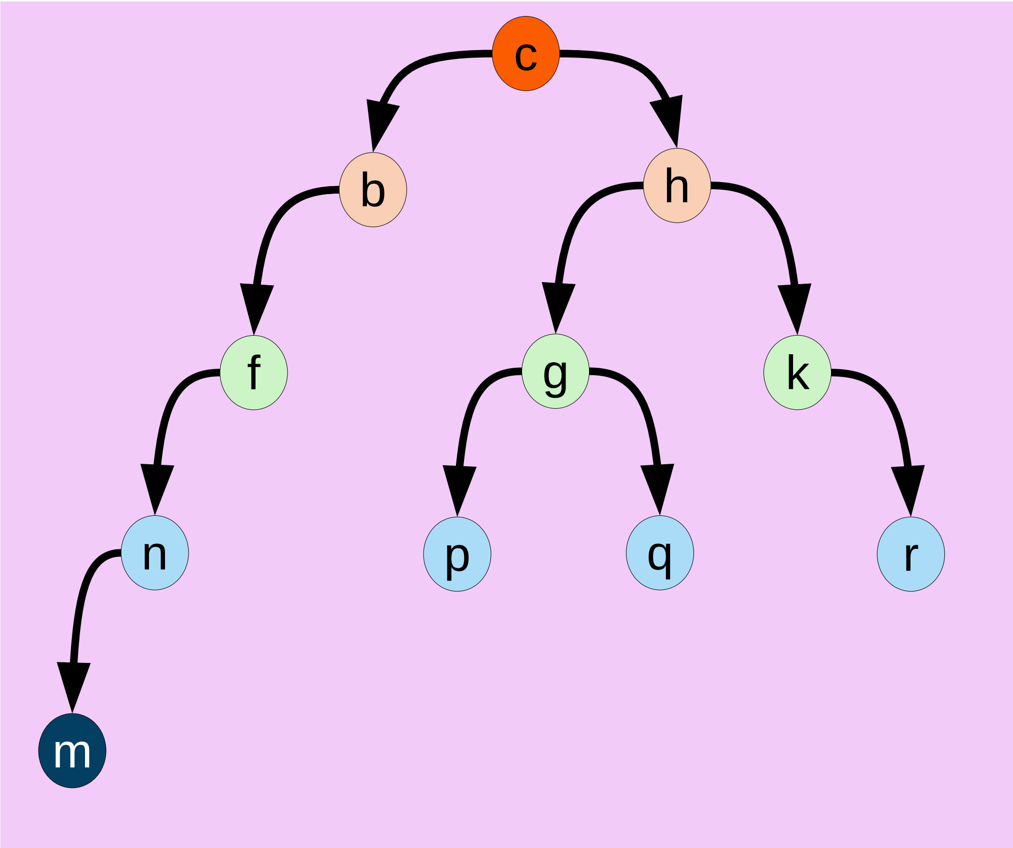
Grafo de entrada

Atividade prática

- Algoritmo de busca em largura iniciando do vértice c



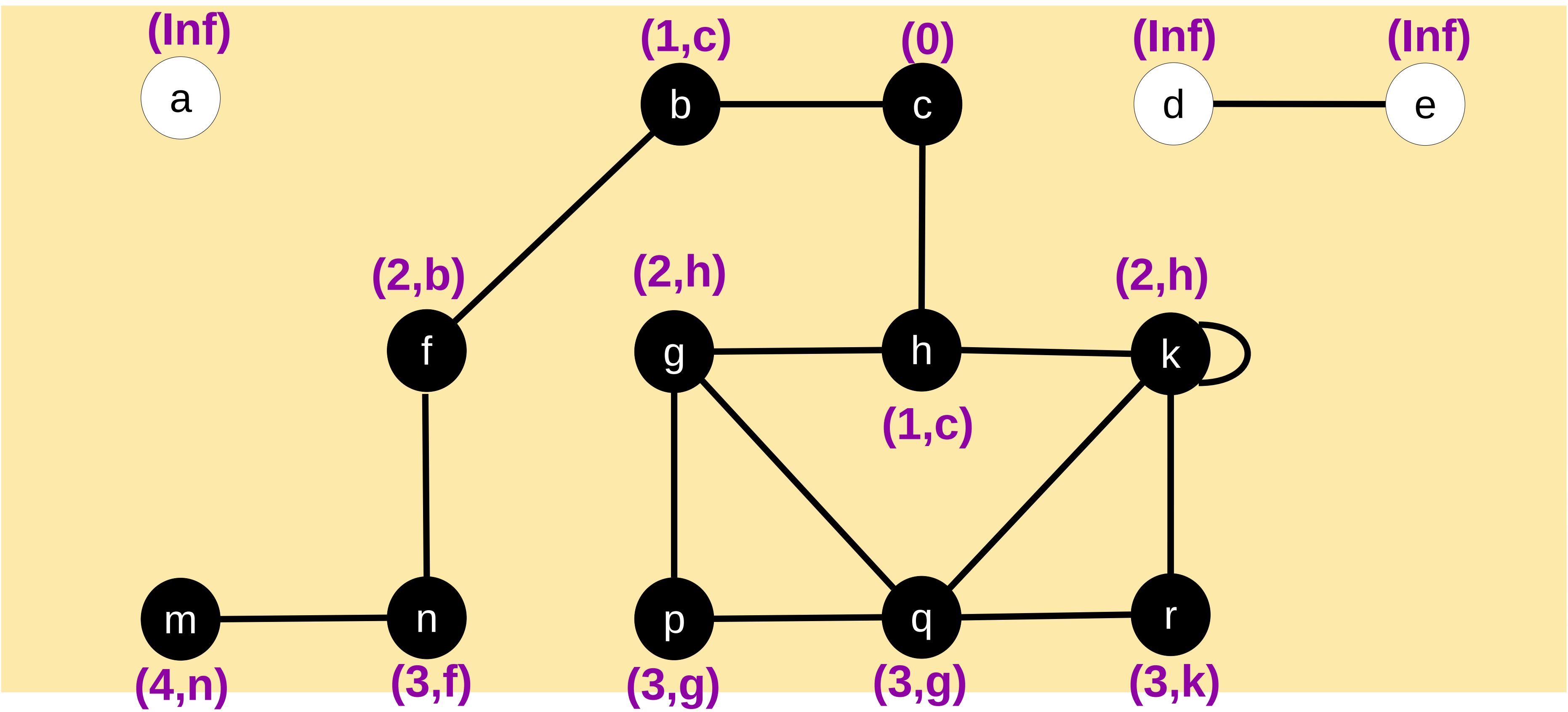
Grafo de entrada



Árvore de busca em largura

Atividade prática

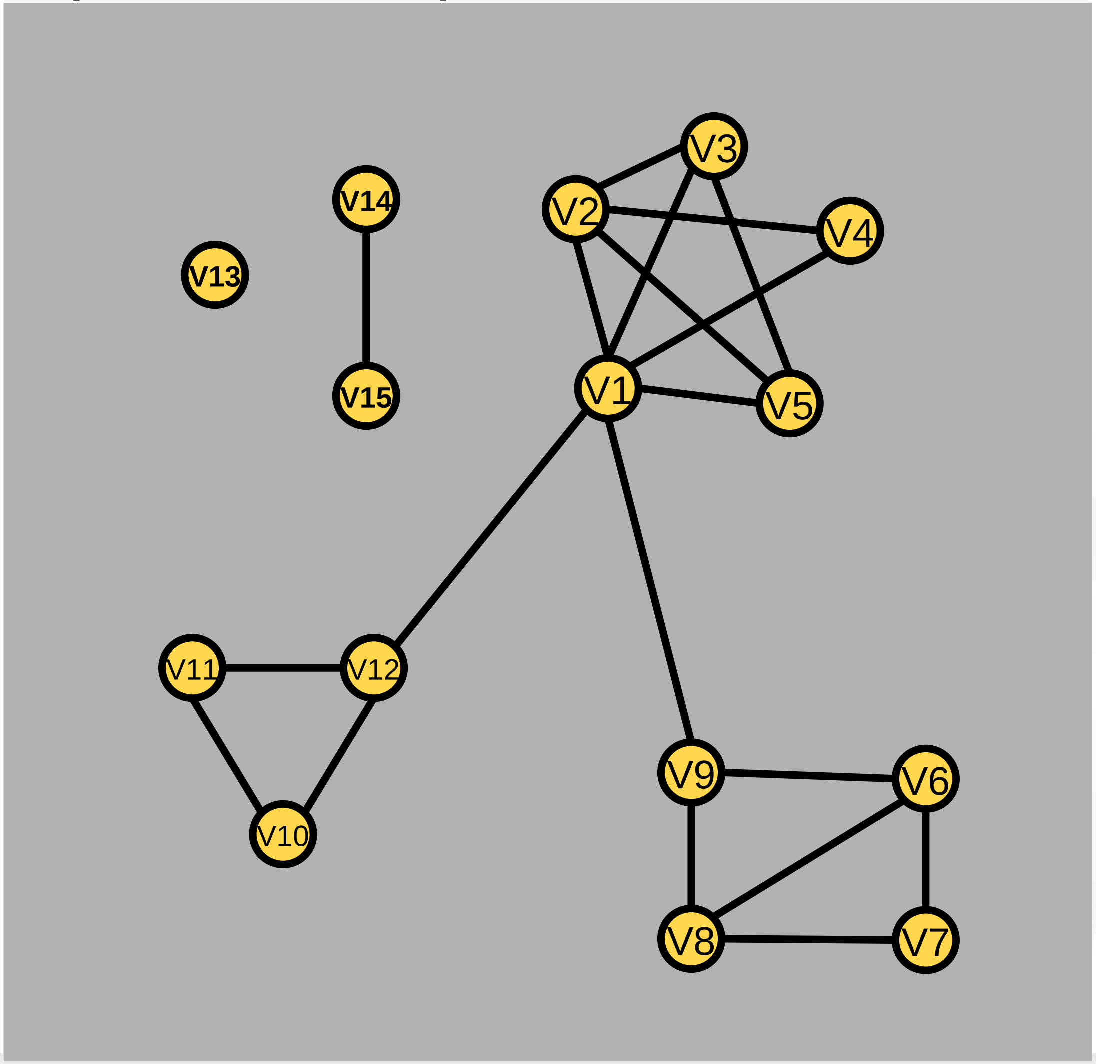
- **1.** Utilize o algoritmo de busca em largura para obter a distância:
  - **b)** Do vértice **c** ao vértice **q.**



Atividade Prática

2. Execute o algoritmo de Busca em Largura a partir do vértice 1 do grafo G. Indique a sequência de vértices visitados, considerando na busca a preferência para vértices de:

- a) menor índice
- b) maior índice

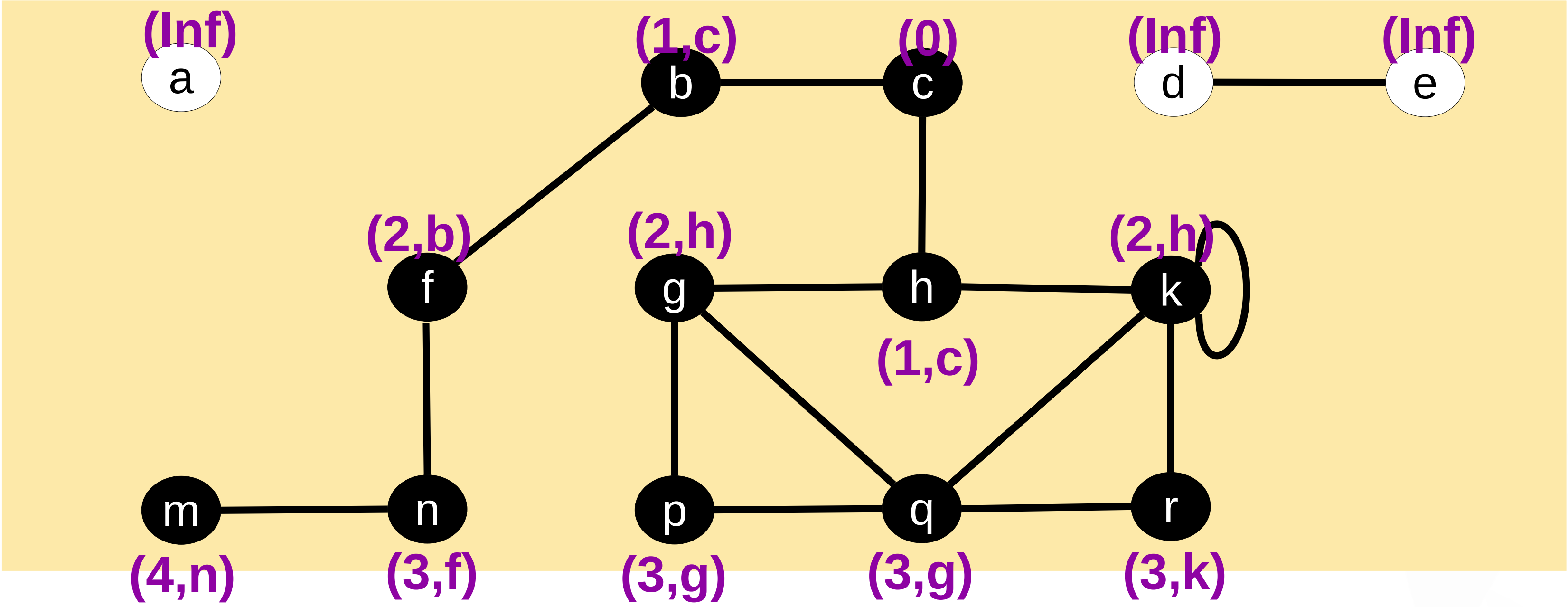


Atividade Prática

2. Execute o algoritmo de Busca em Largura a partir do vértice 1 do grafo G. Indique a sequência de vértices visitados, considerando na busca a preferência para vértices de:

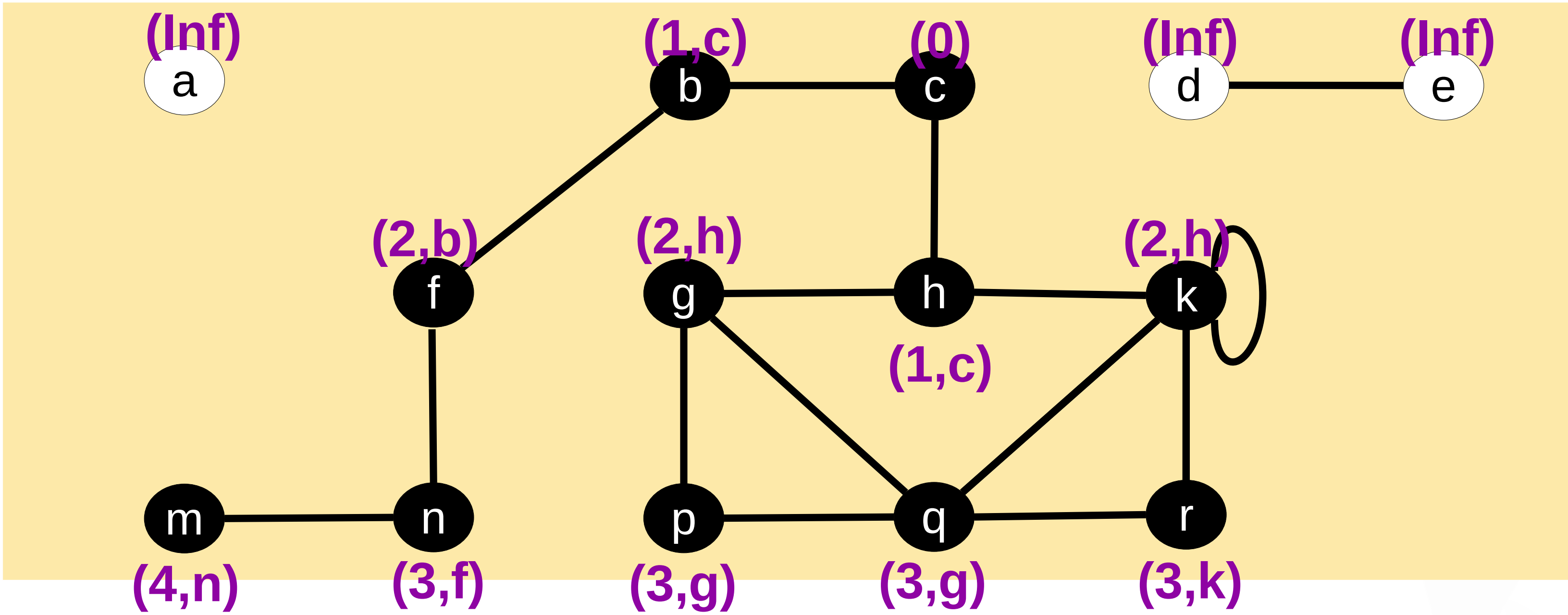
(a) menor índice	**1,2,3,4,5,9,12,6,8,10,11,7**
(b) maior índice	**1,12,9,5,4,3,2,11,10,8,6,7**

Para finalizar...



- **Como verificar se o grafo é não-conexo?**
- **Como saber quais vértices são alcançáveis a partir de c?**

Para finalizar...



- **Como verificar se o grafo é não-conexo?**

  Examinando o atributo v.dist: se pelo menos um deles for igual a INFINITO, o grafo não é conexo

- **Como saber quais vértices são alcançáveis a partir de c?**

  Examinando o atributo v.dist. Um vértice v é alcançável se:

  - v.dist $\neq$ INFINITO.