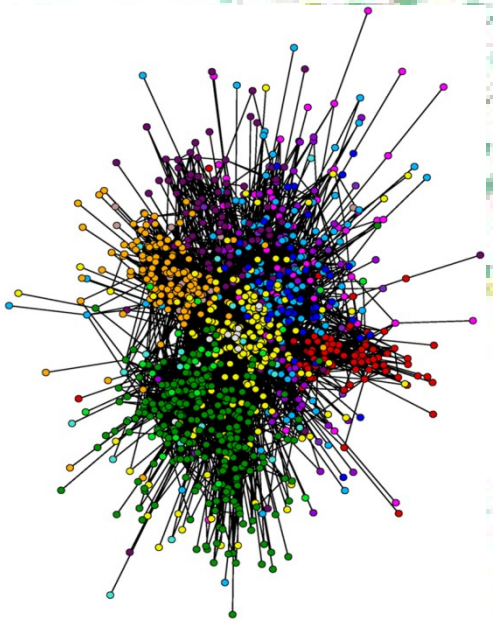
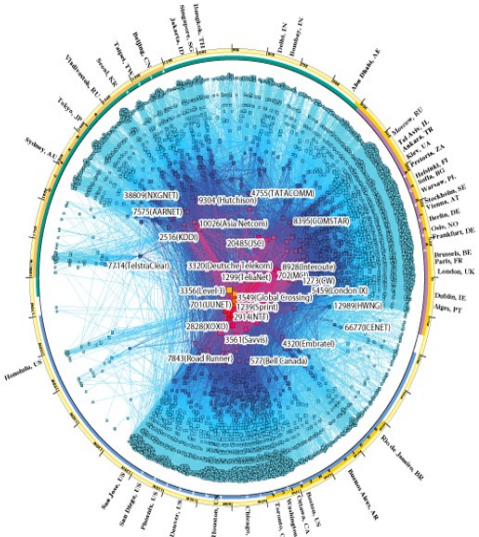
Aula 4 – Busca em Profundidade





Aula passada

- Algoritmo de busca em largura.

Busca em profundidade

Busca em profundidade

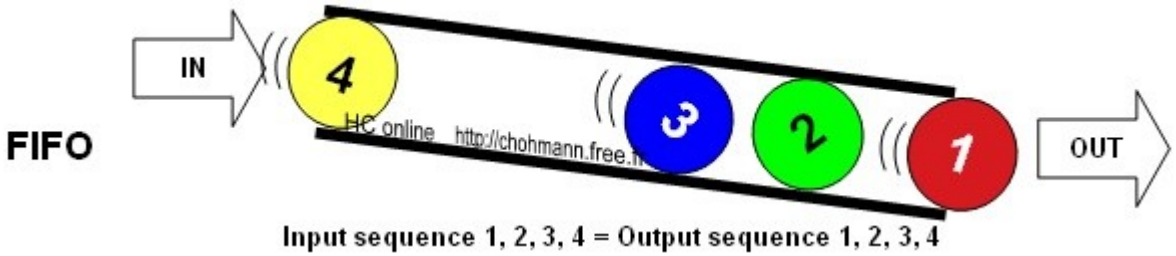
- A estrategia seguida pela busca em profundidade é, como seu nome implica, **procurar cada vez mais “fundo” no grafo**.

- Nessa busca as arestas são **exploradas a partir do vértice mais recentemente descoberto** que ainda possui arestas inexploradas saindo dele.

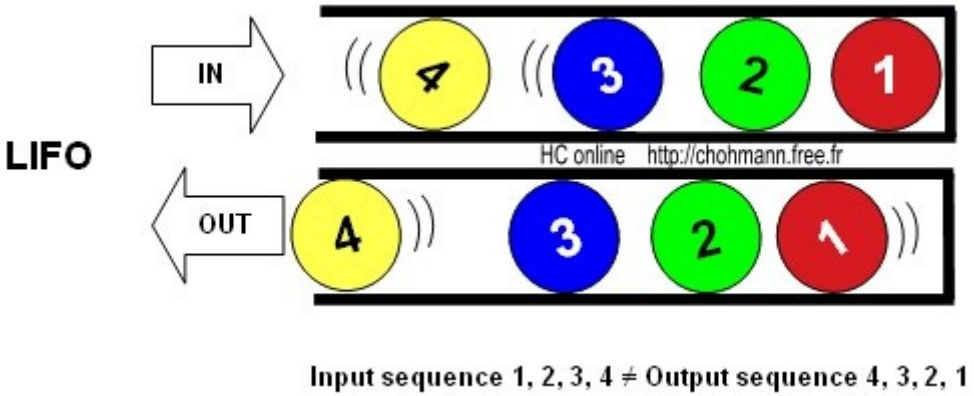
- Baseado em **Pilha**.

Busca em profundidade

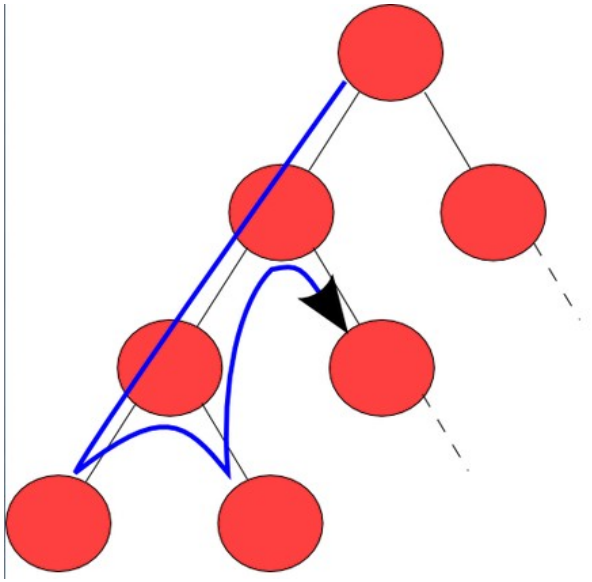
Fila



Pilha

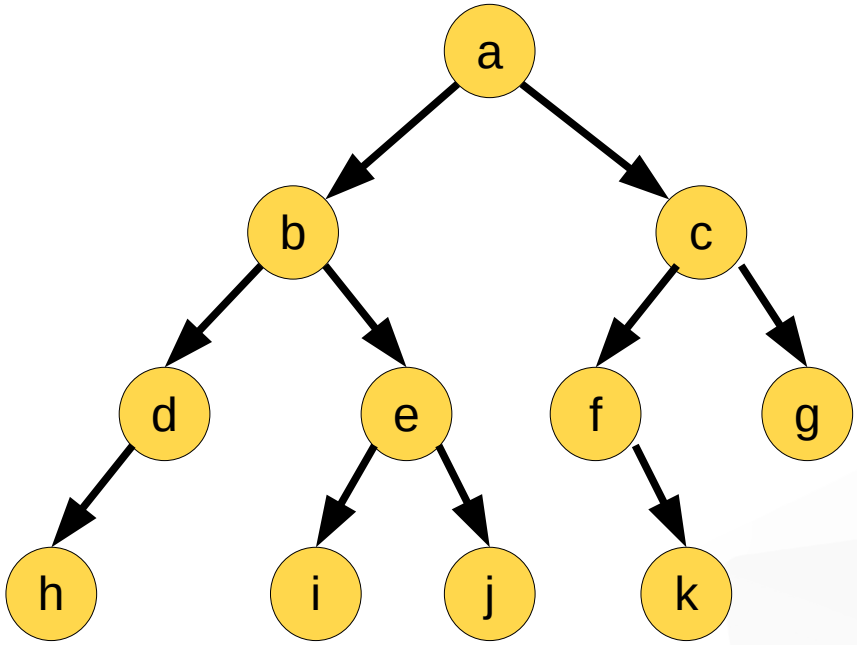


Busca em profundidade

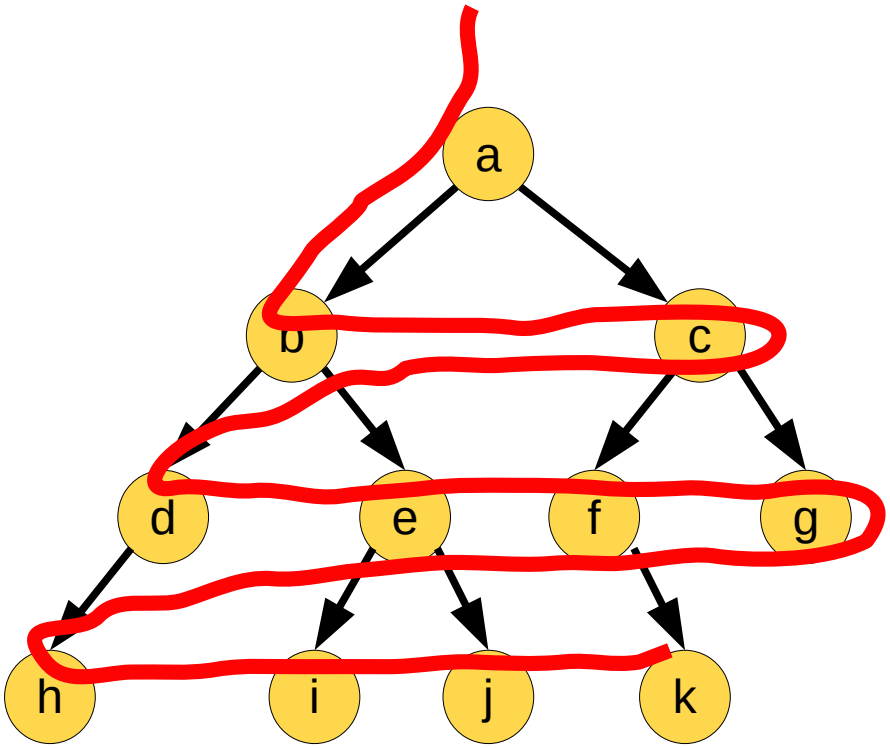


- Inicialmente **{s}**.
- Os vértices são explorados a partir do vértice mais recentemente descoberto.

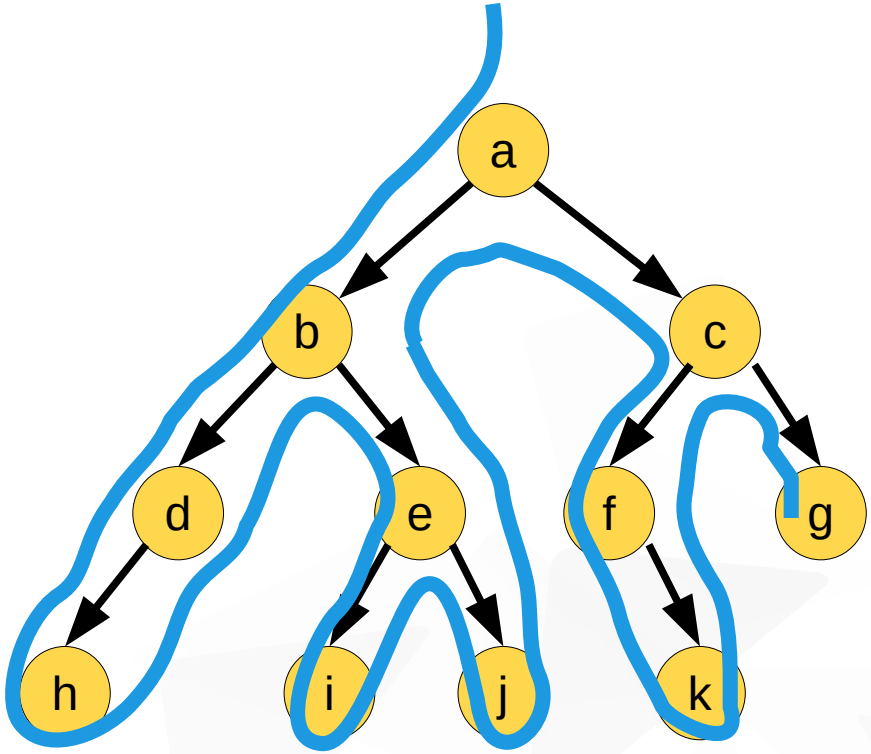
Busca em grafos



Busca em grafos



Busca em largura
(Breadth First Searh - **BFS**)



Busca em profundidade
(Depth First Searh - **DFS**)

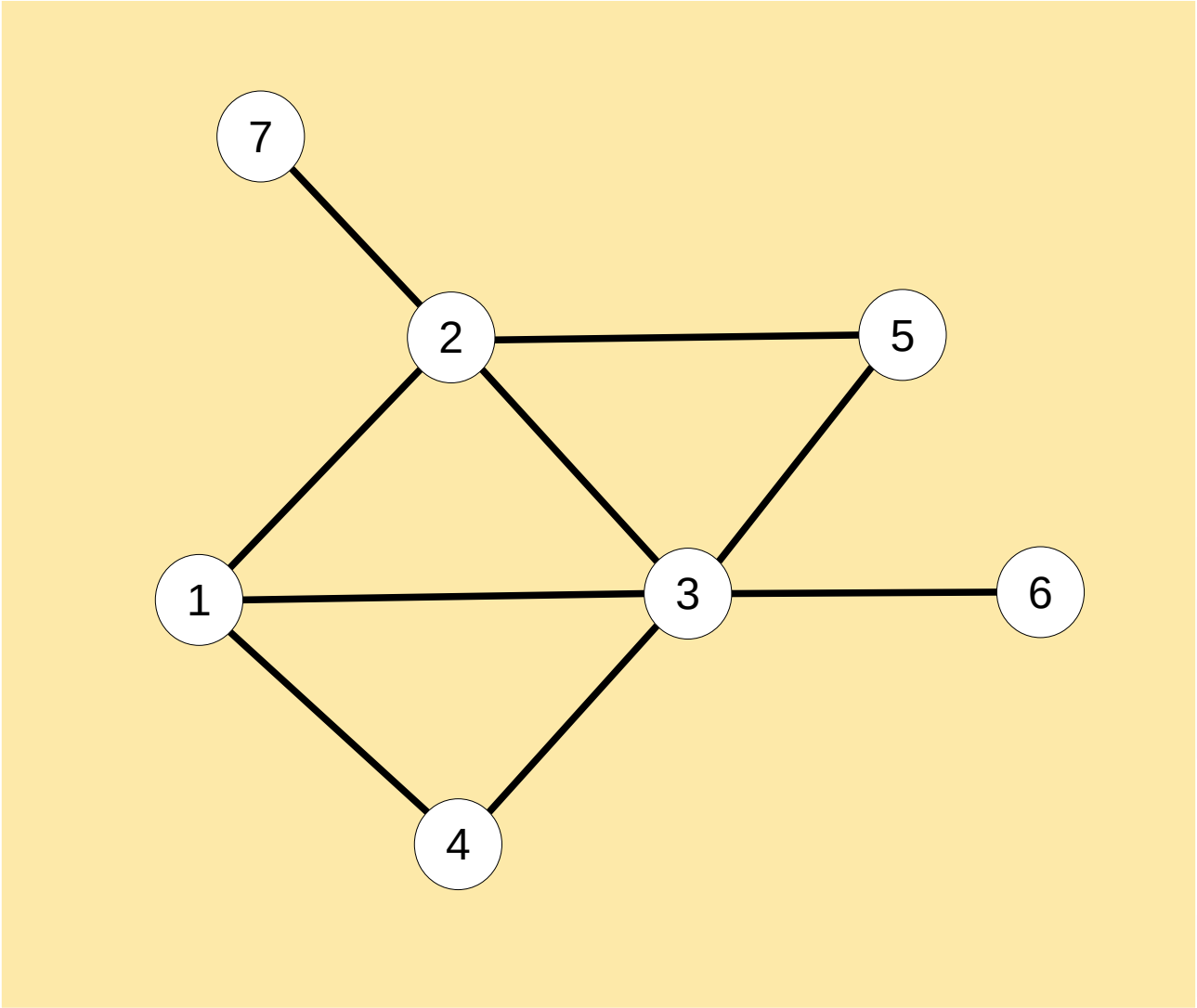
Busca em profundidade

O algoritmo de busca em profundidade também **atribuirá cores** a cada vértice

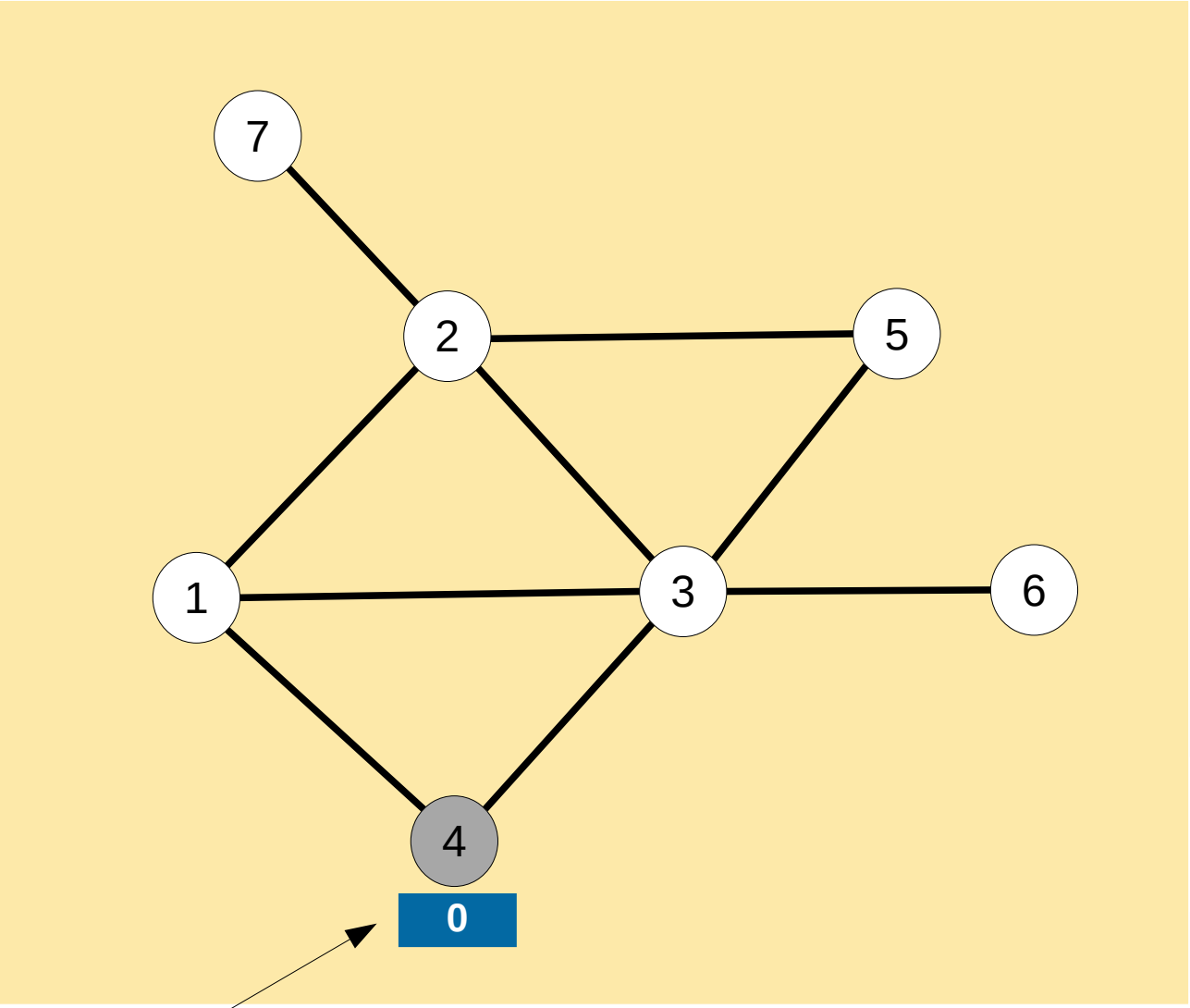
- Cor branca = "não visitado". Inicialmente todos os vértices são brancos.
- Cor cinza = "visitado pela primeira vez".
- Cor preta = "teve seus vizinhos visitados".

Busca em profundidade

Grafo inicial

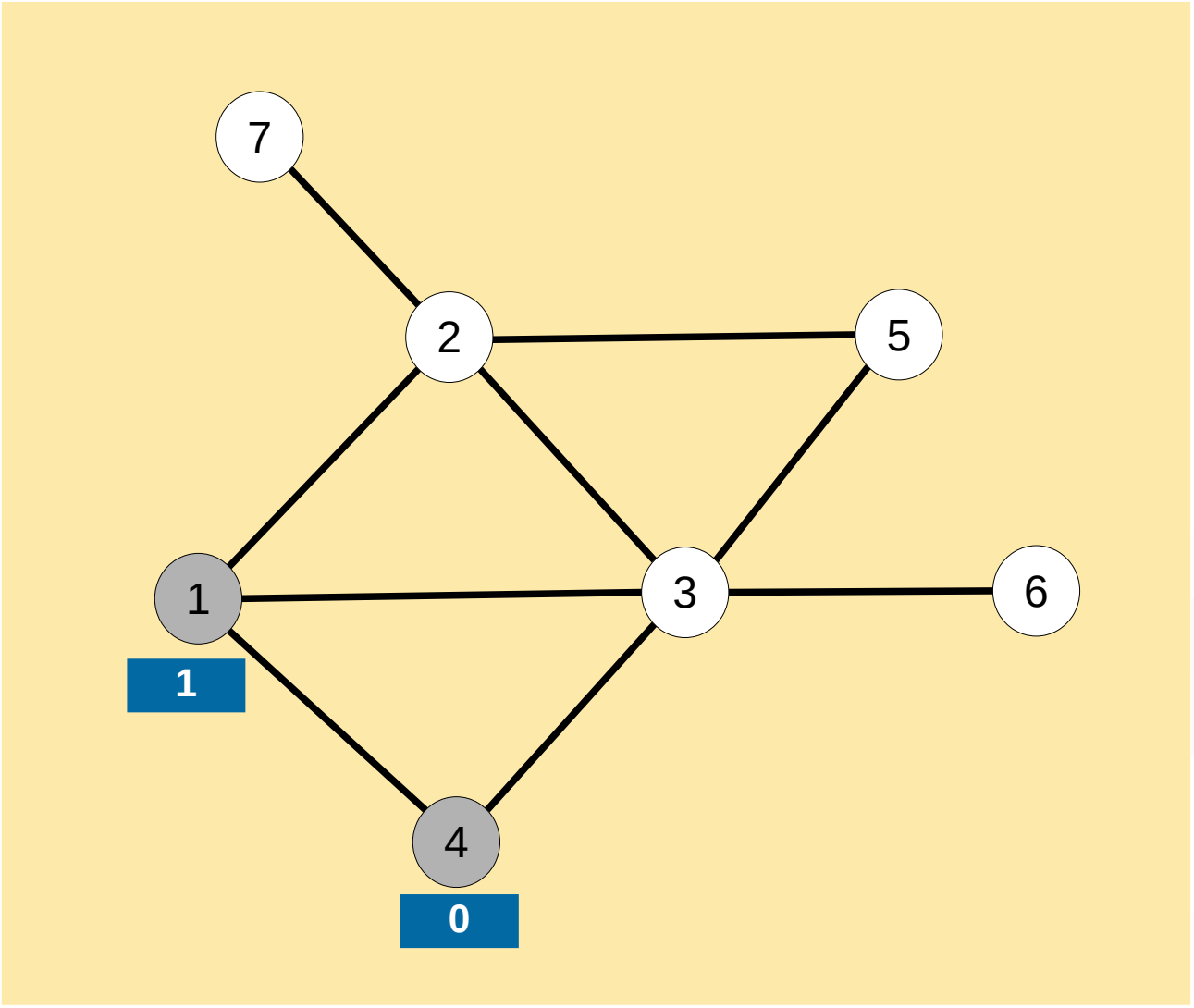


Busca em profundidade (origem s=4)



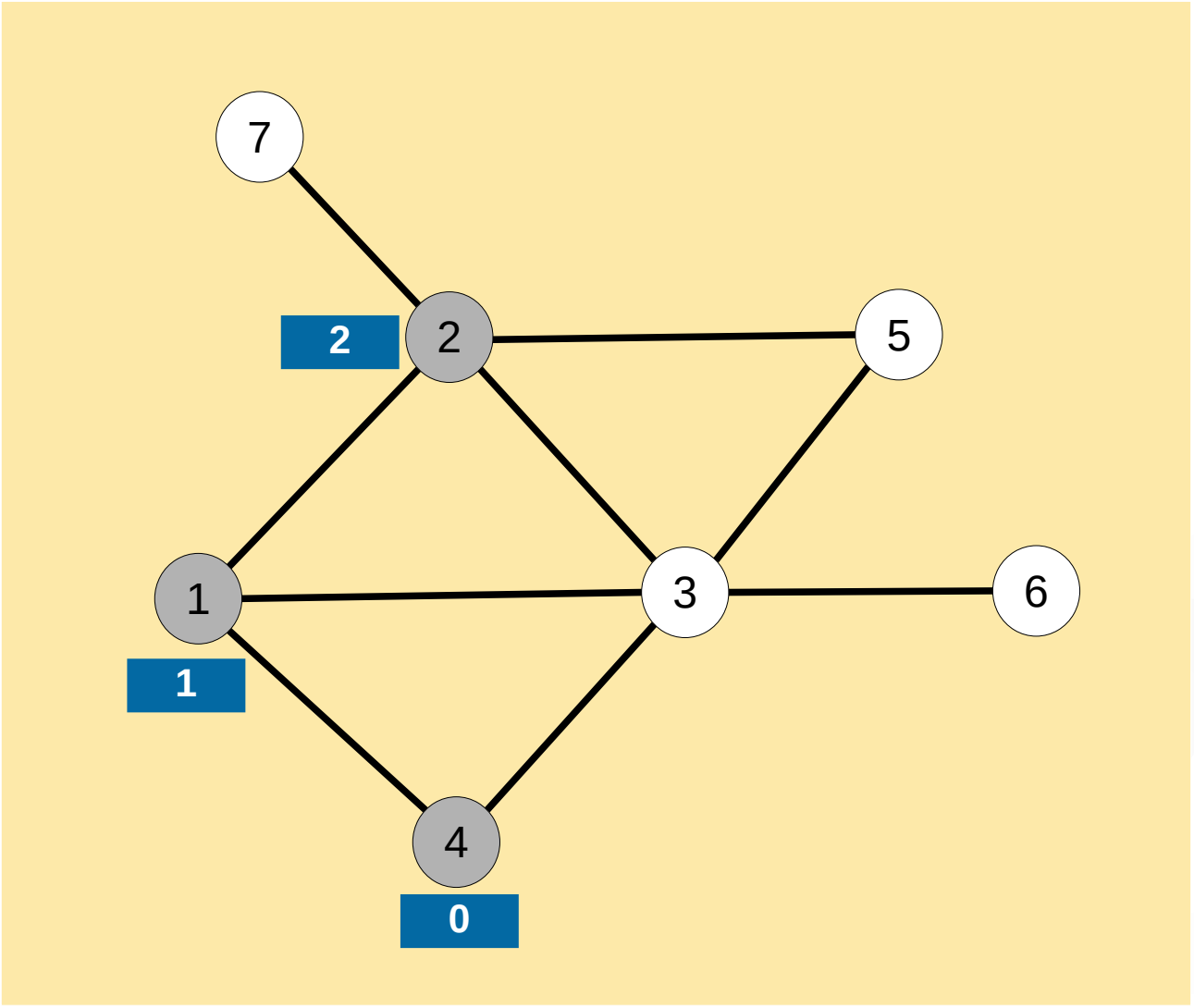
Pilha={4}

Busca em profundidade (origem s=4)



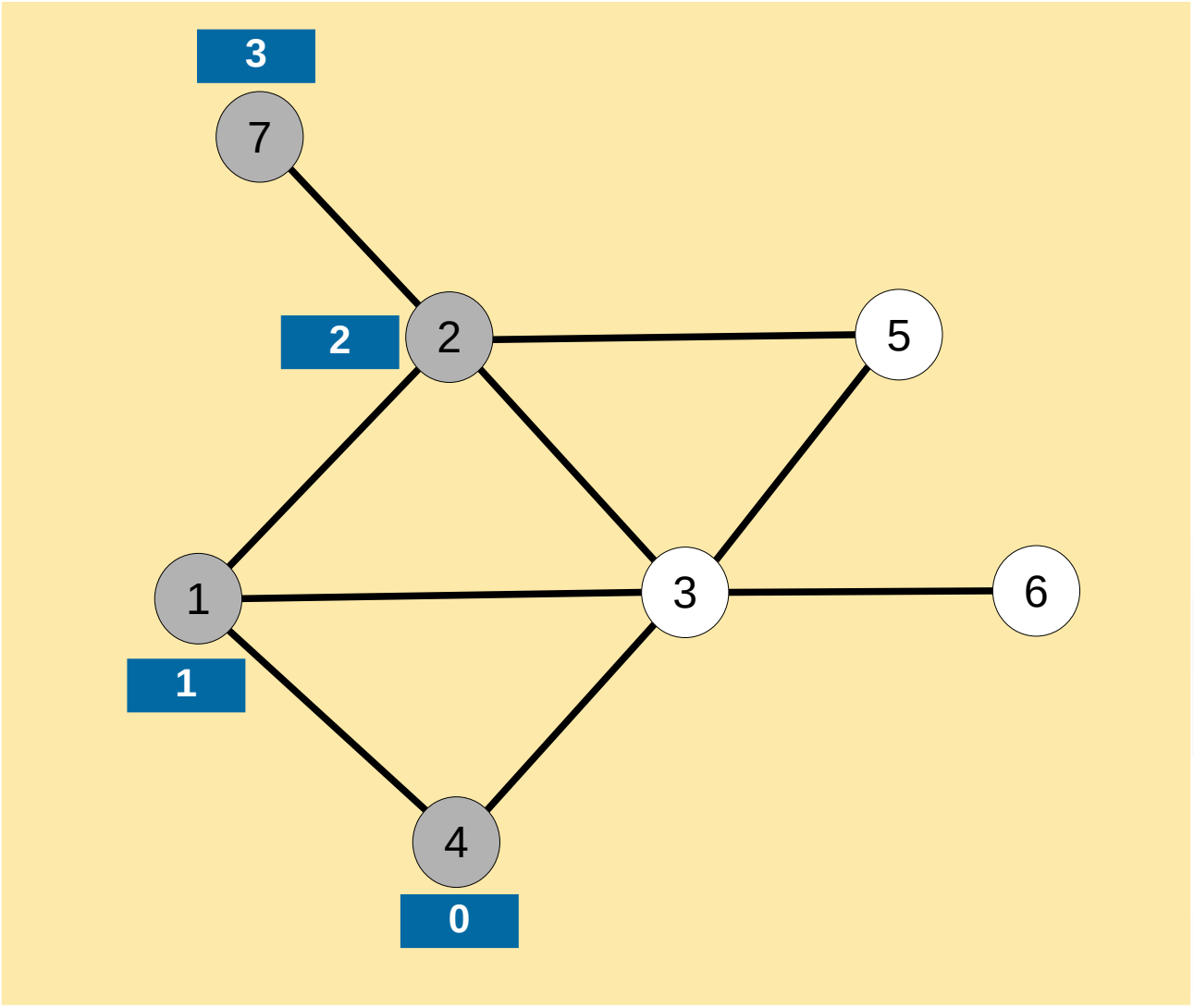
Pilha={4,1}

Busca em profundidade (origem s=4)



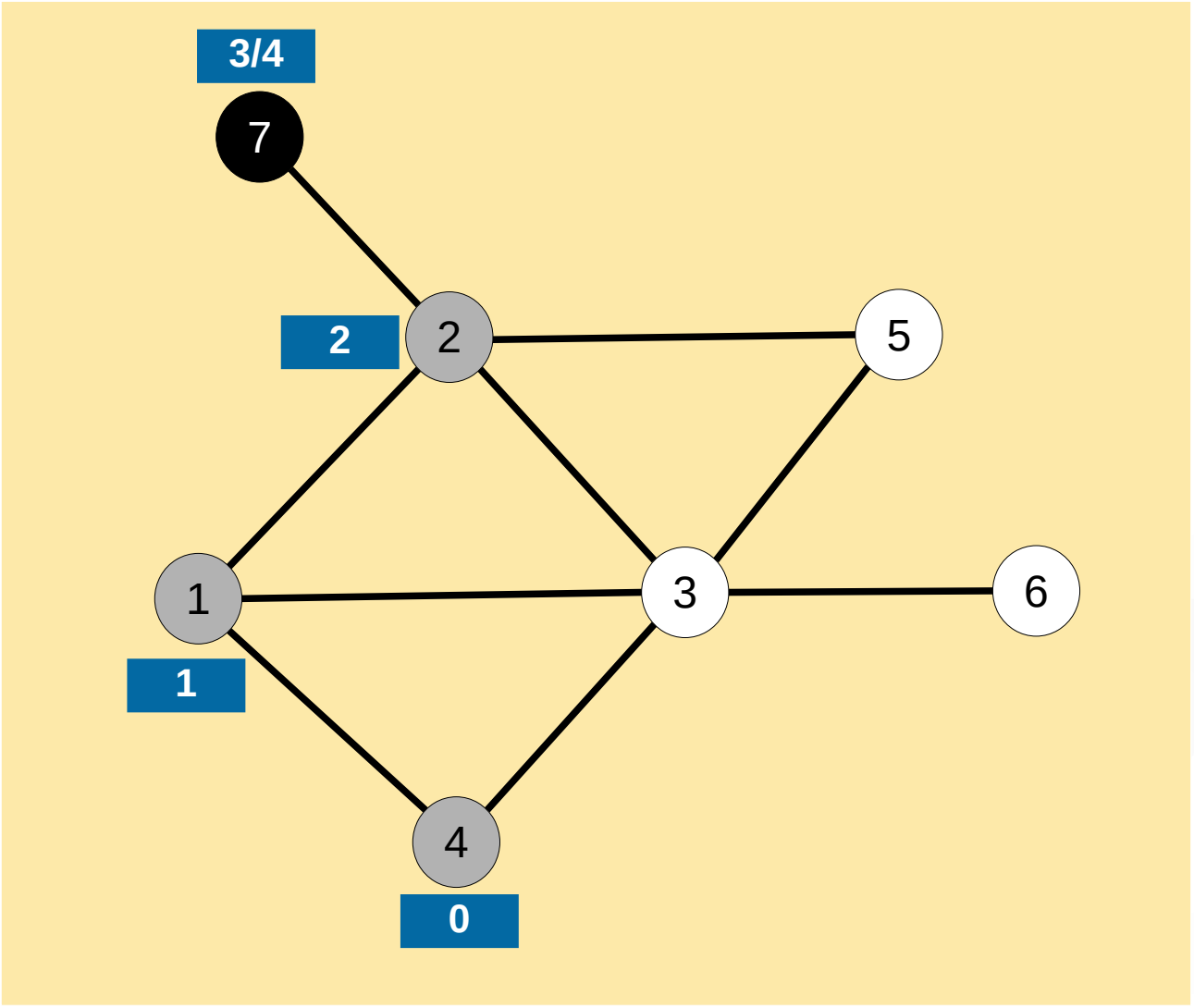
Pilha={4,1,2}

Busca em profundidade (origem s=4)



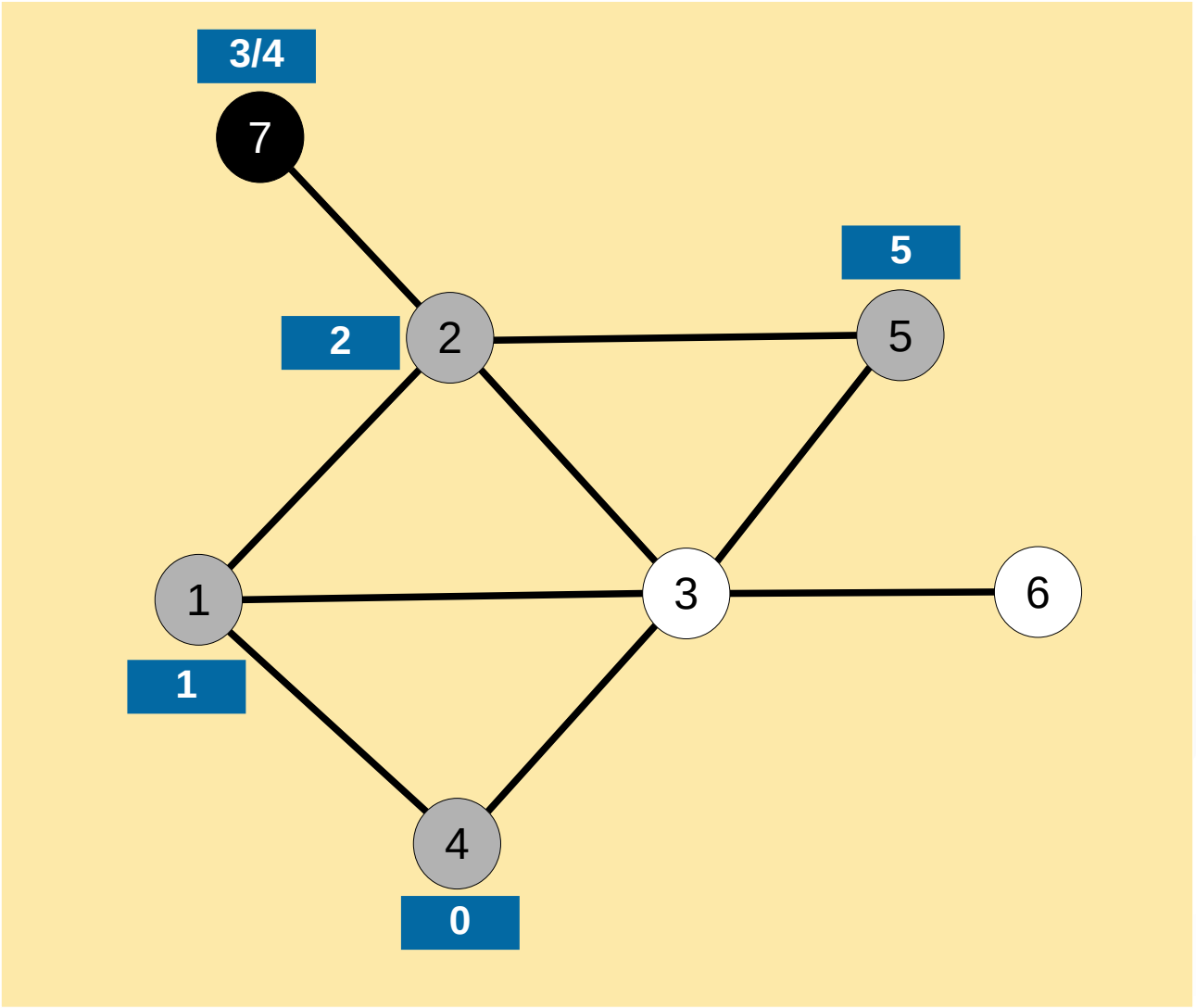
Pilha={4,1,2,7}

Busca em profundidade (origem s=4)



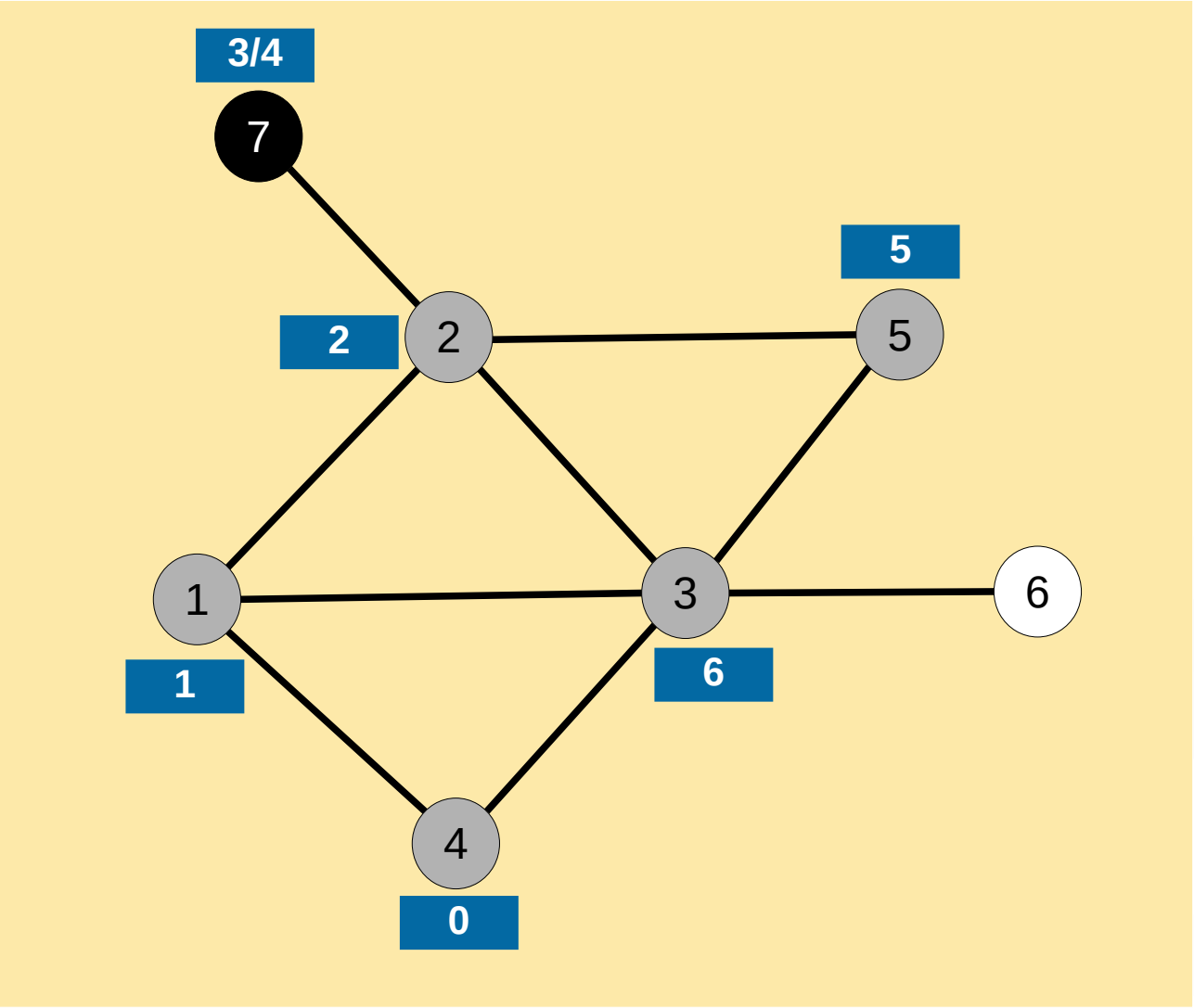
Pilha={4,1,2}

Busca em profundidade (origem s=4)



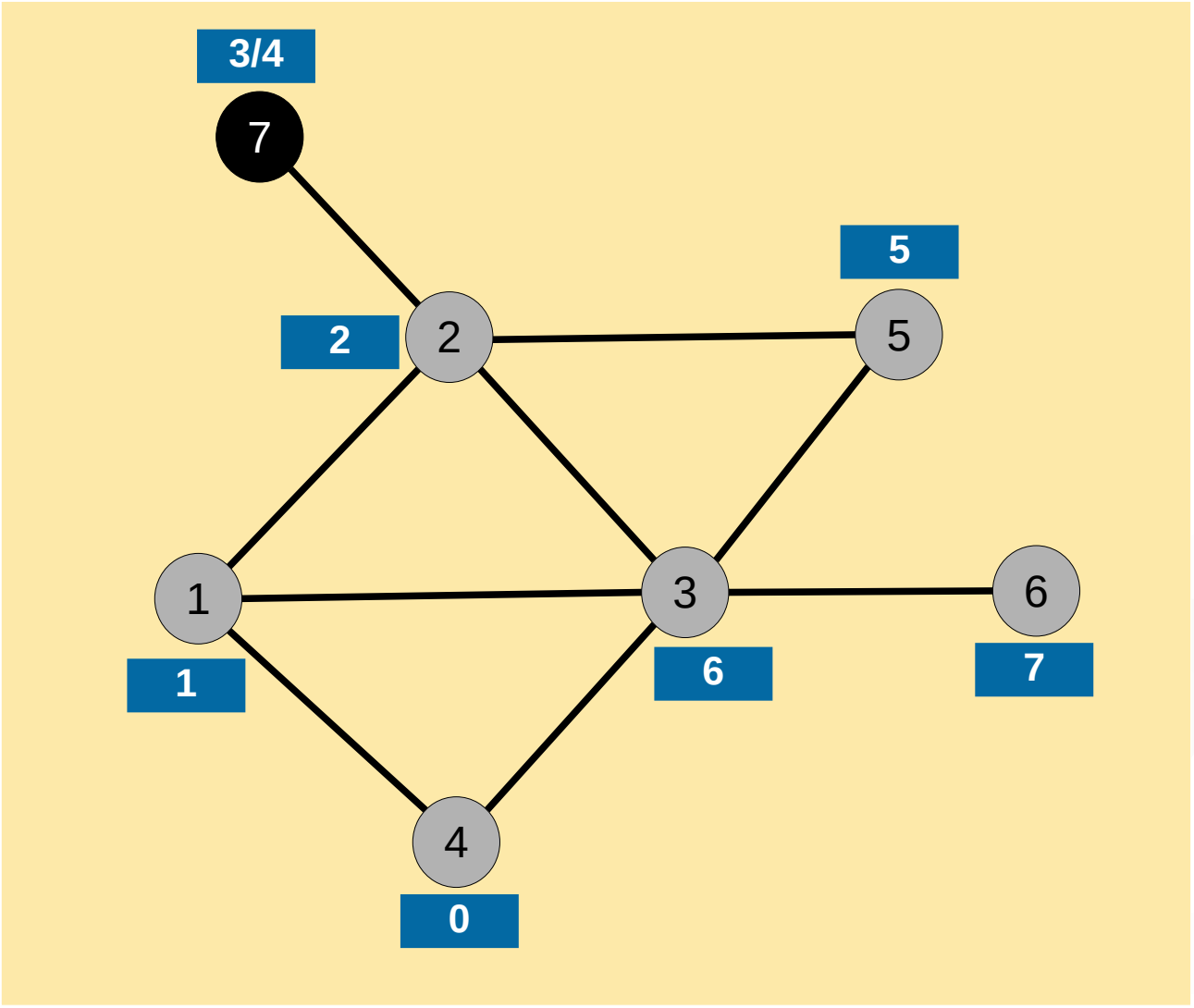
Pilha={4,1,2,5}

Busca em profundidade (origem s=4)



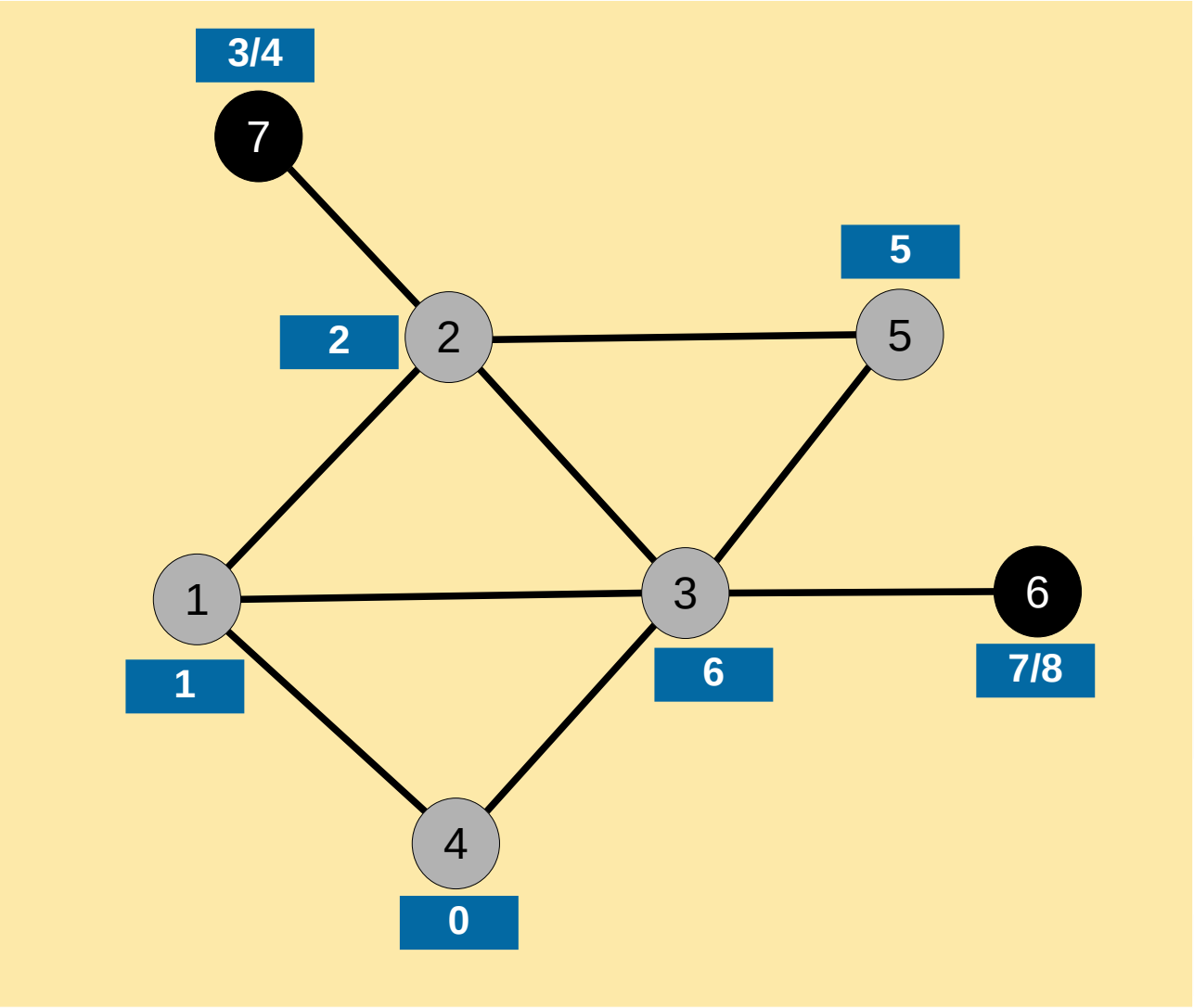
Pilha={4,1,2,5,3}

Busca em profundidade (origem s=4)



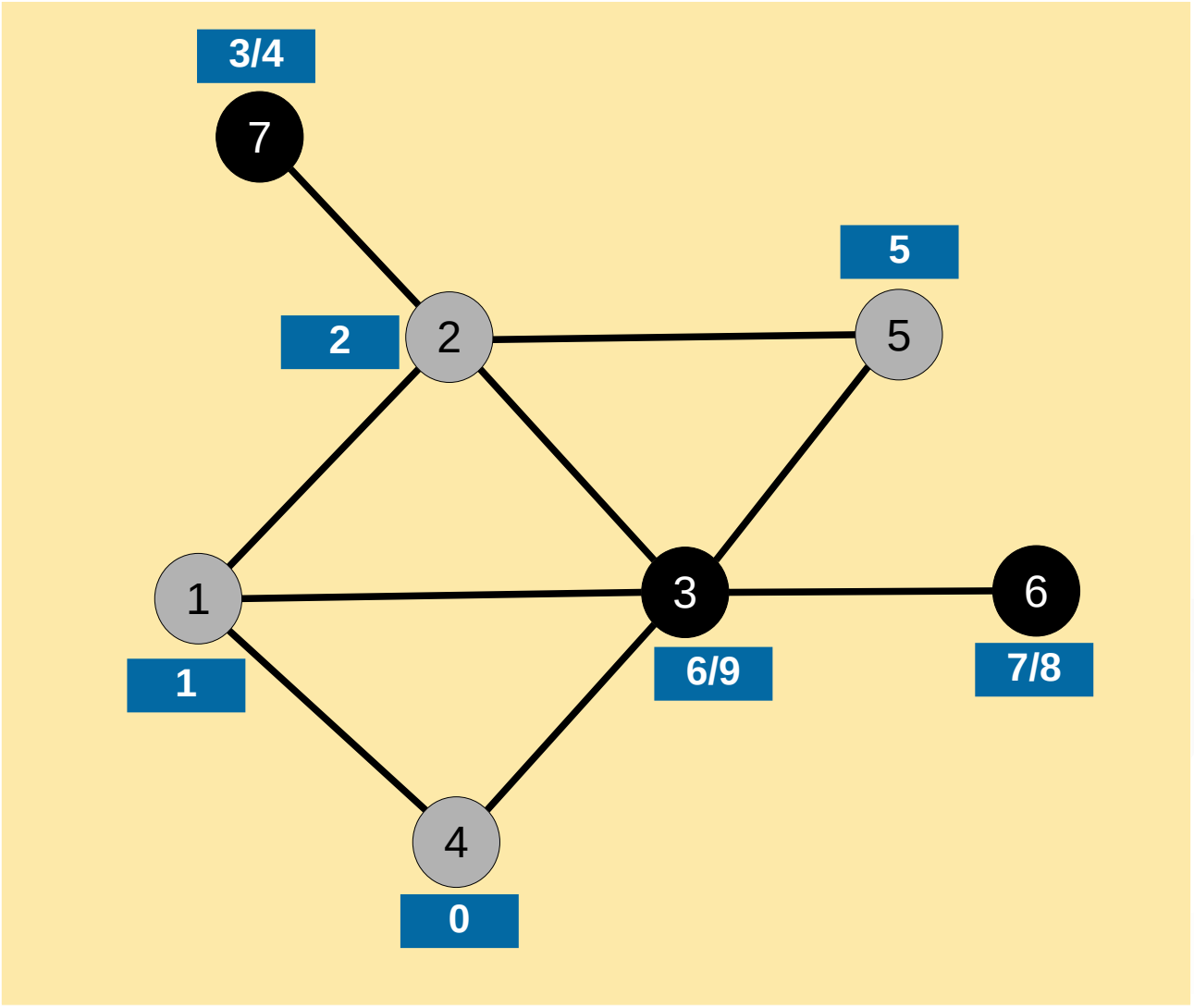
Pilha={4,1,2,5,3,6}

Busca em profundidade (origem s=4)



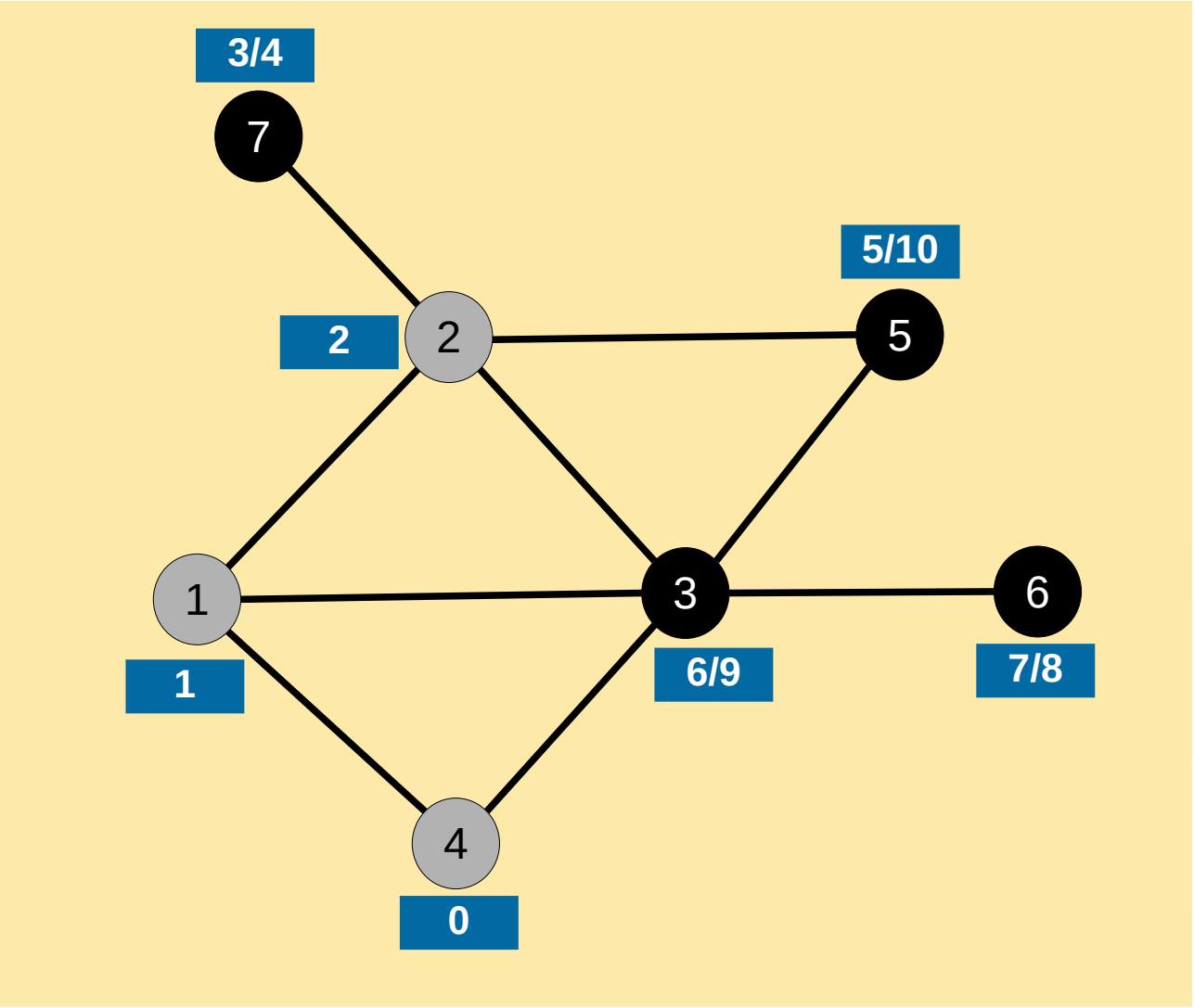
Pilha={4,1,2,5,3}

Busca em profundidade (origem s=4)



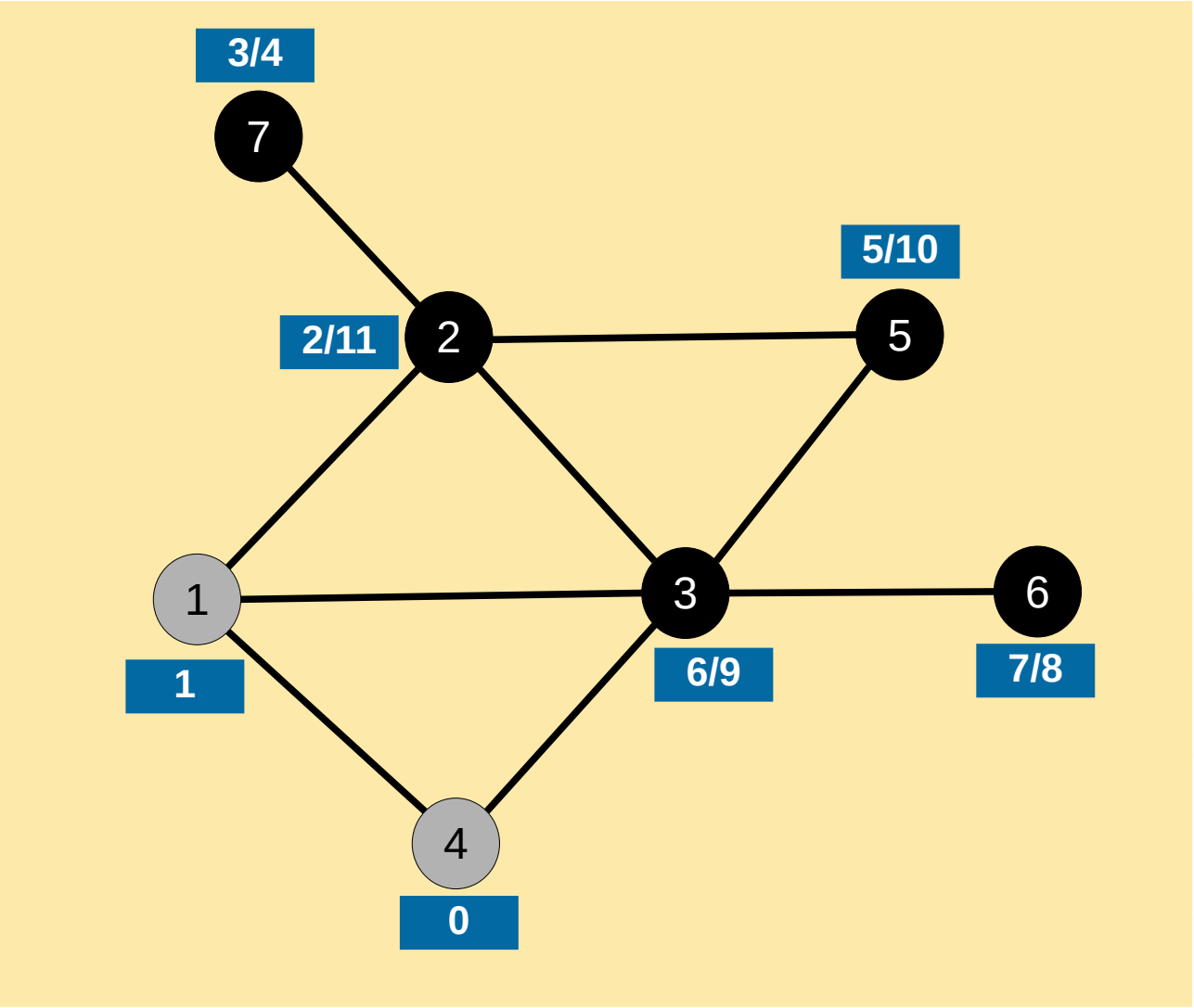
Pilha={4,1,2,5}

Busca em profundidade (origem s=4)



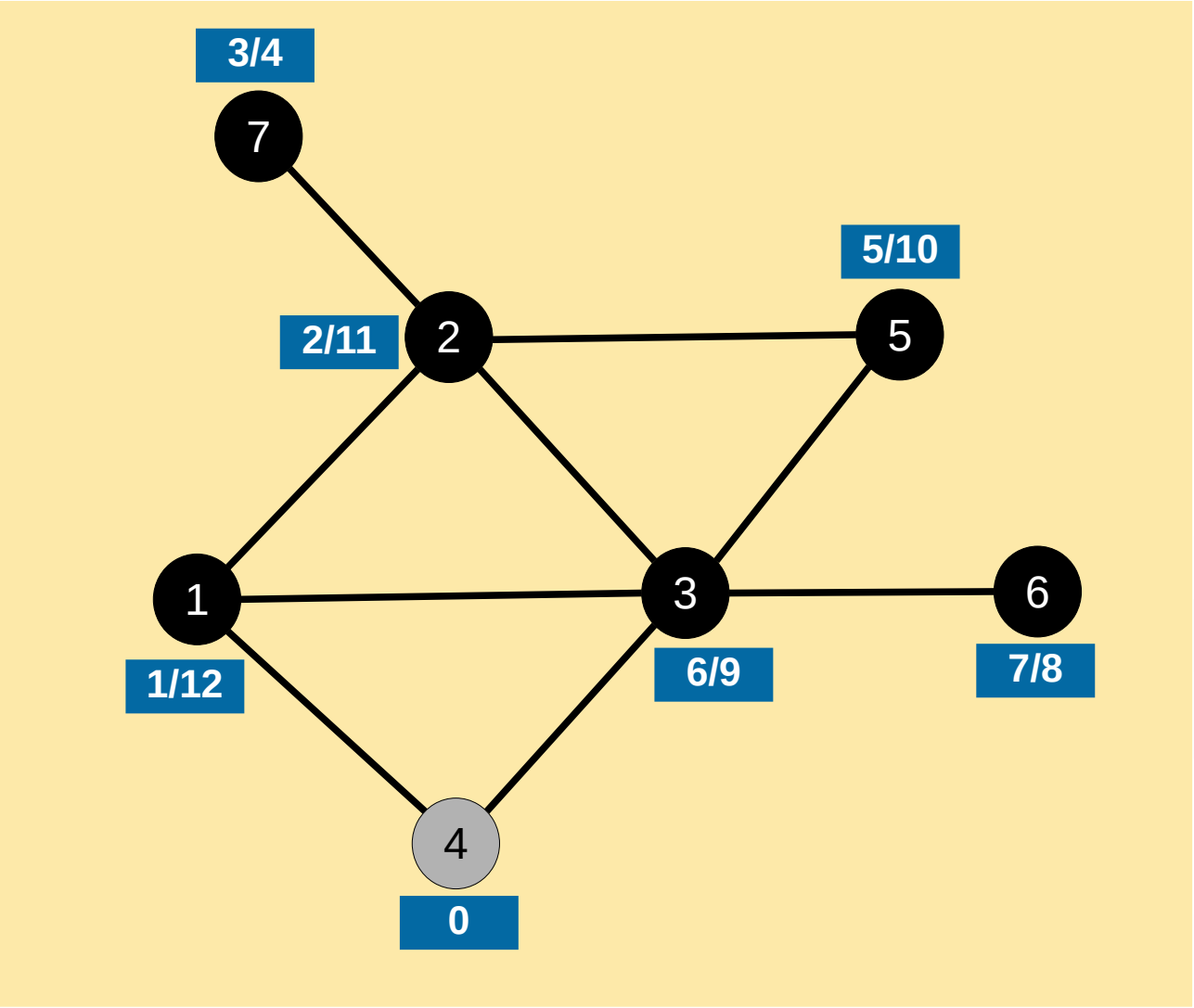
Pilha={4,1,2}

Busca em profundidade (origem s=4)



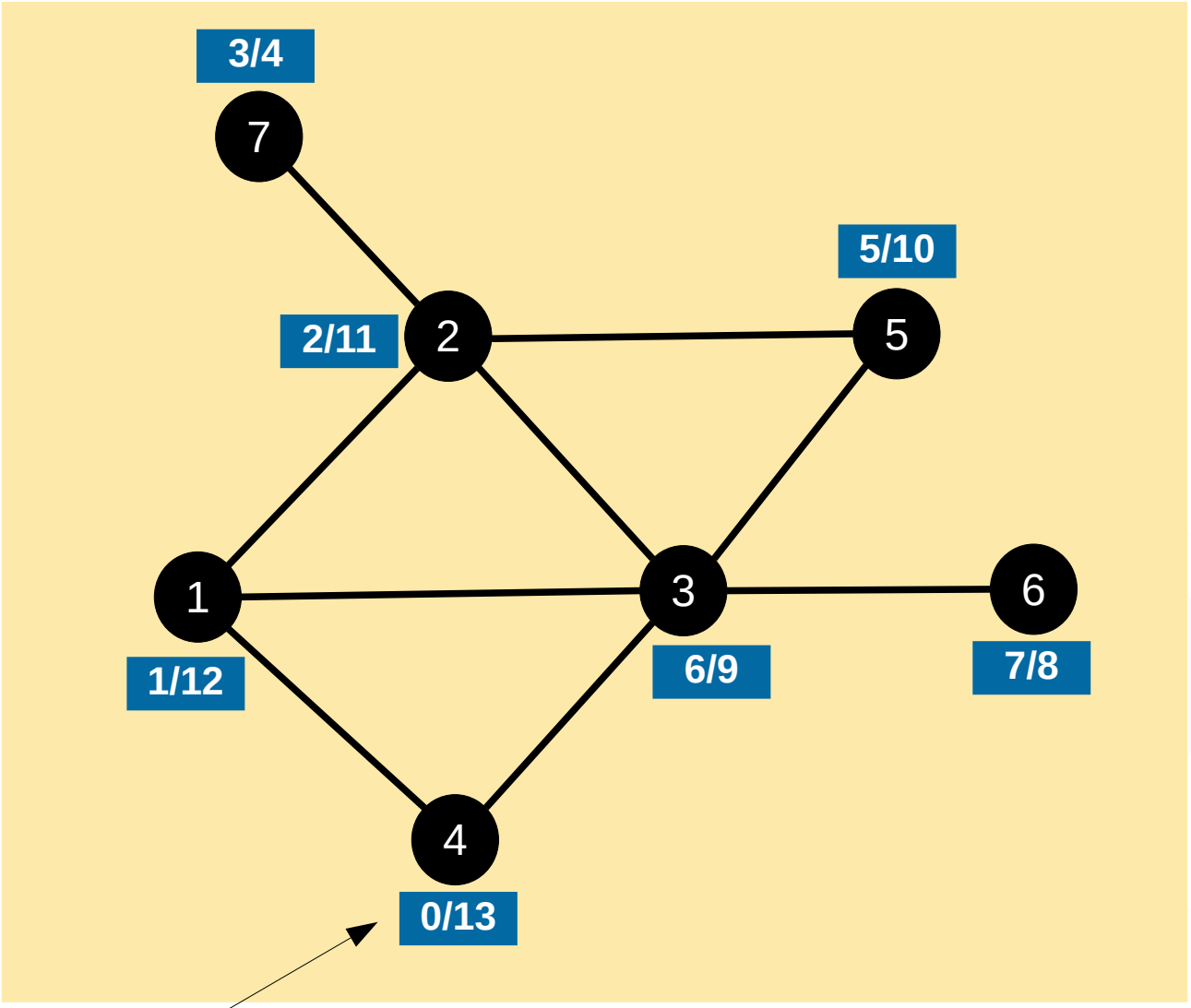
Pilha={4,1}

Busca em profundidade (origem s=4)



Pilha={4}

Busca em profundidade (origem s=4)



Pilha={}

Busca em profundidade (Algoritmo)

Constantes:

- BRANCO, CINZA, PRETO, INFINITO

Variáveis:

- P (Pilha) , s (vértice de origem)

Propriedades do vértice v:

- v.cor (cor do vértice)
- v.t1 (tempo de visita inicial do vértice **v**)
- v.t2 (tempo de visita final do vértice **v**)

Funções :

- Insere(P,v), permite inserir o vértice **v** na pilha **P**.
- Remove(P), permite remover um vértice da pilha **P**.
- Consulta(P), permite consultar o último vértice na pilha **P**.

Busca em profundidade (Depth First Search)

```
DFS(G,s):
  Para cada vértice v em G.V-{s} faça
      v.cor = BRANCO
      v.t1  = INFINITO
      v.t2  = INFINITO
  tempo = 0
  s.cor = CINZA
  s.t1  = tempo
  P = VAZIO
  Insere(P,s)

  Enquanto P ≠ VAZIO faça
      u = Topo(P)
      Se u tem pelo menos um vértice adjacente BRANCO
          v = escolhe um dos vértices adjacentes com v.cor=BRANCO
          v.cor = CINZA
          tempo = tempo+1
          v.t1  = tempo
          Insere(P,v)
      Caso-contrário
          u.cor = PRETO
          tempo = tempo+1
          v.t2  = tempo
          Remove(P)
```

Busca em profundidade (Depth First Search)

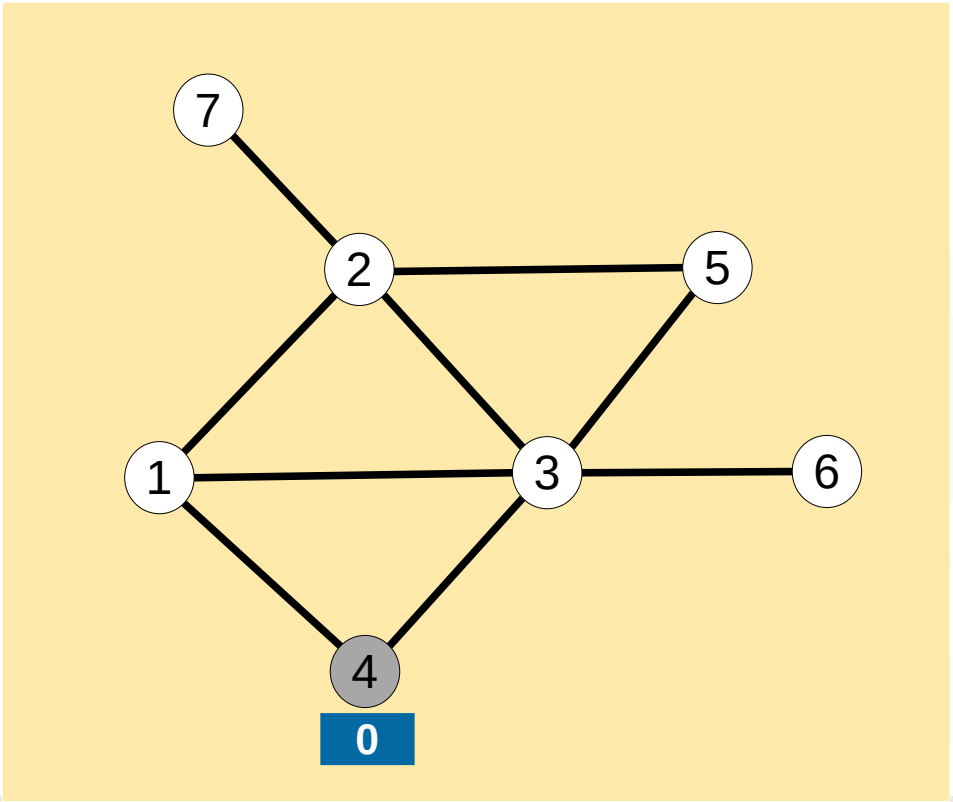
```
DFS(G,s):
  Para cada vértice v em G.V-{s} faça                    Inicialização
      v.cor = BRANCO
      v.t1  = INFINITO
      v.t2  = INFINITO
  tempo = 0
  s.cor = CINZA
  s.t1  = tempo
  P = VAZIO
  Insere(P,s)

  Enquanto P ≠ VAZIO faça                                Percorre o grafo
      u = Topo(P)
      Se u tem pelo menos um vértice adjacente BRANCO
          v = escolhe um dos vértices adjacentes com v.cor=BRANCO
          v.cor = CINZA
          tempo = tempo+1
          v.t1  = tempo
          Insere(P,v)
      Caso-contrário
          u.cor = PRETO
          tempo = tempo+1
          v.t2  = tempo
          Remove(P)
```

Busca em profundidade (Depth First Search)

```
Para cada vértice v em G.V-{s} faça
    v.cor = BRANCO
    v.t1  = INFINITO
    v.t2  = INFINITO
tempo = 0
s.cor = CINZA
s.t1  = tempo
P = VAZIO
Insere(P,s)
```

Inicialização

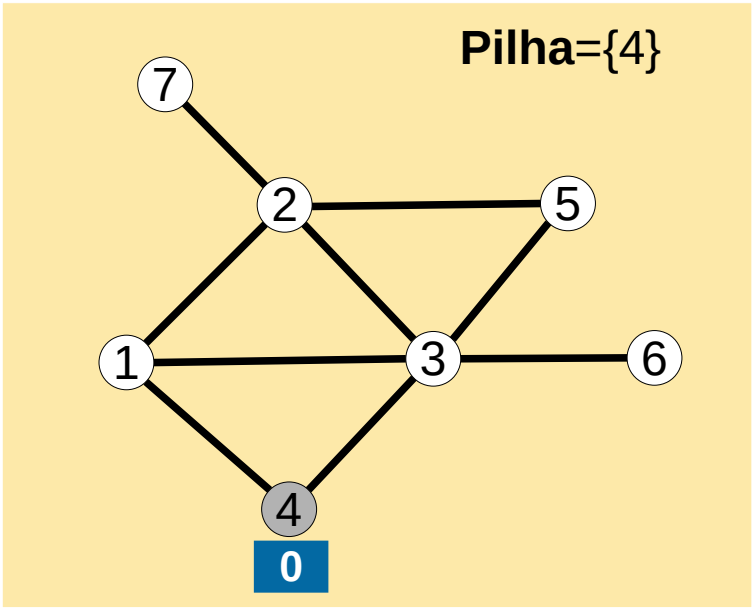


P={4}

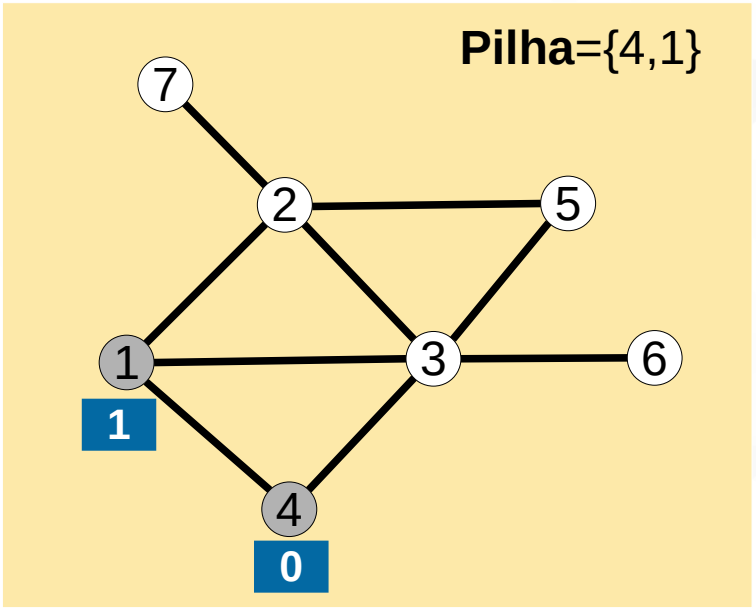
Busca em profundidade (Depth First Search)

Percorre o grafo

```
Enquanto P ≠ VAZIO faça
    u = Topo(P)
    Se u tem pelo menos um vértice adjacente BRANCO
        v = escolhe um dos vértices adjacentes com v.cor=BRANCO
        v.cor = CINZA
        tempo = tempo+1
        v.t1  = tempo
        Insere(P,v)
    Caso-contrário
        u.cor = PRETO
        tempo = tempo+1
        v.t2  = tempo
        Remove(P)
```



Inicialização



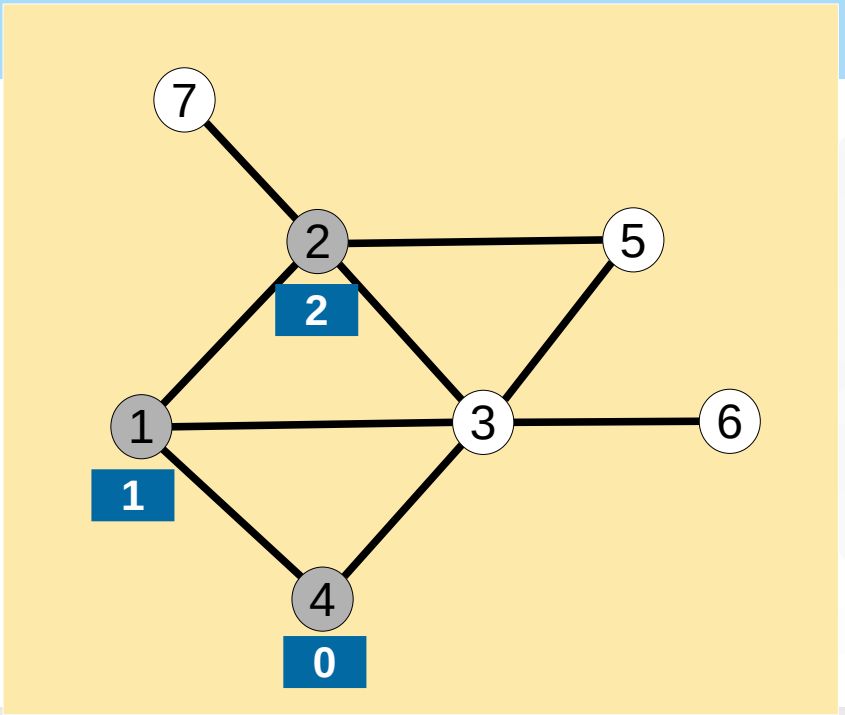
Iteração 1

Busca em profundidade (Depth First Search)

Percorre o grafo

```
Enquanto P ≠ VAZIO faça
    u = Topo(P)
    Se u tem pelo menos um vértice adjacente BRANCO
        v = escolhe um dos vértices adjacentes com v.cor=BRANCO
        v.cor = CINZA
        tempo = tempo+1
        v.t1  = tempo
        Insere(P,v)
    Caso-contrário
        u.cor = PRETO
        tempo = tempo+1
        v.t2  = tempo
        Remove(P)
```

Iteração 2



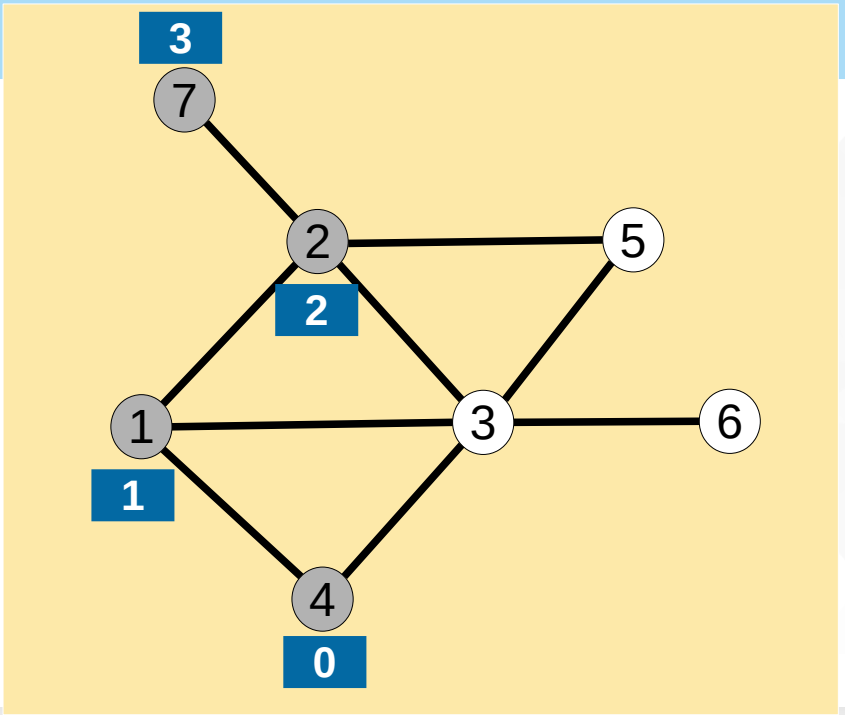
Pilha={4,1,2}

Busca em profundidade (Depth First Search)

Percorre o grafo

```
Enquanto P ≠ VAZIO faça
    u = Topo(P)
    Se u tem pelo menos um vértice adjacente BRANCO
        v = escolhe um dos vértices adjacentes com v.cor=BRANCO
        v.cor = CINZA
        tempo = tempo+1
        v.t1  = tempo
        Insere(P,v)
    Caso-contrário
        u.cor = PRETO
        tempo = tempo+1
        v.t2  = tempo
        Remove(P)
```

Iteração 3



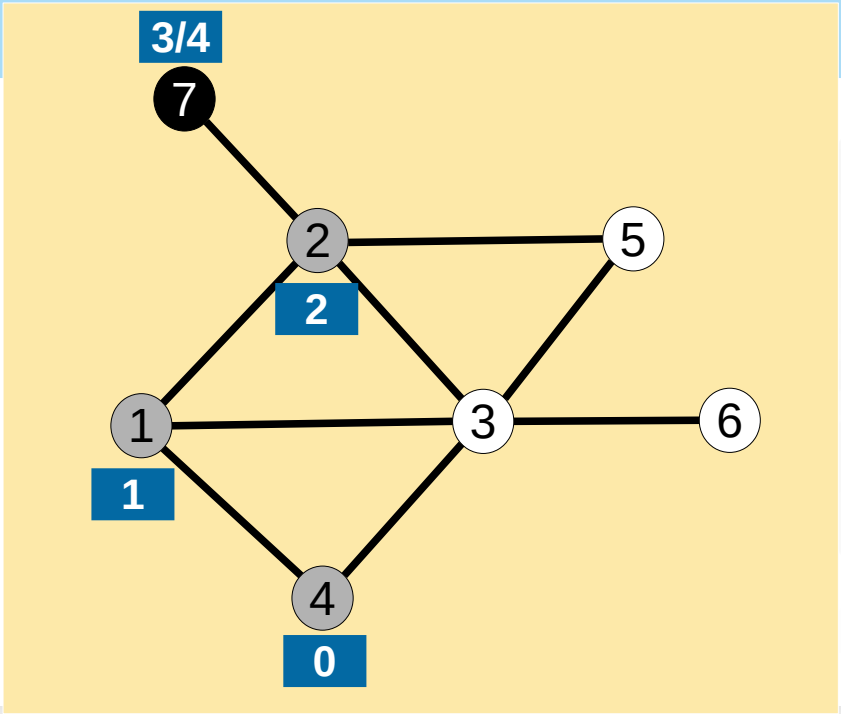
Pilha={4,1,2,7}

Busca em profundidade (Depth First Search)

Percorre o grafo

```
Enquanto P ≠ VAZIO faça
    u = Topo(P)
    Se u tem pelo menos um vértice adjacente BRANCO
        v = escolhe um dos vértices adjacentes com v.cor=BRANCO
        v.cor = CINZA
        tempo = tempo+1
        v.t1  = tempo
        Insere(P,v)
    Caso-contrário
        u.cor = PRETO
        tempo = tempo+1
        v.t2  = tempo
        Remove(P)
```

Iteração 4



Pilha={4,1,2}

Busca em profundidade

- O algoritmo anterior pode ser transformado a uma versão **RECURSIVA.**

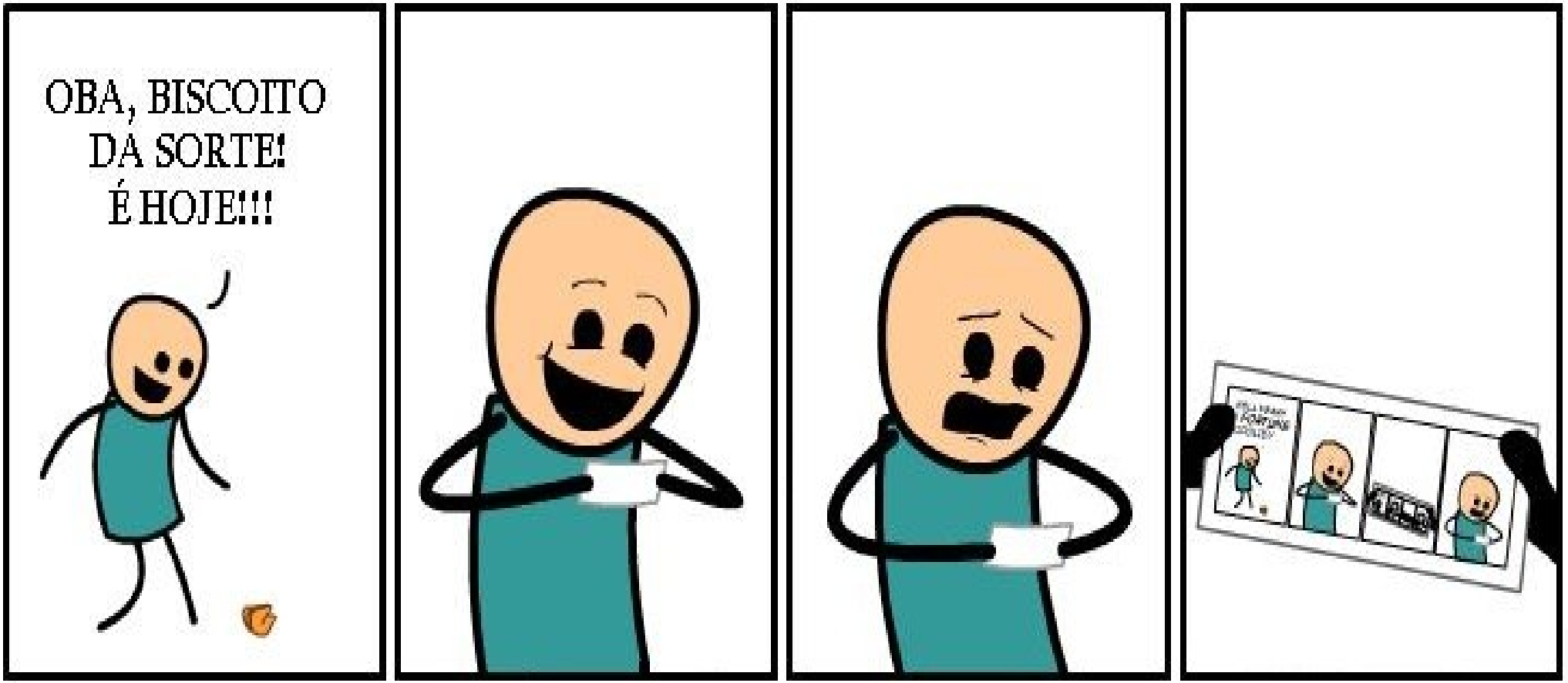
  Dado um grafo **G** e um vértice **v**, seja percorrido todo o grafo G usando a **busca em profundidade**.

- O algoritmo **recursivo**, é uma variante que **simplifica o uso da estrutura de pilha** (P).

Recursividade

Uma função (programa) recursivo é uma função que se “chama a si mesma”.

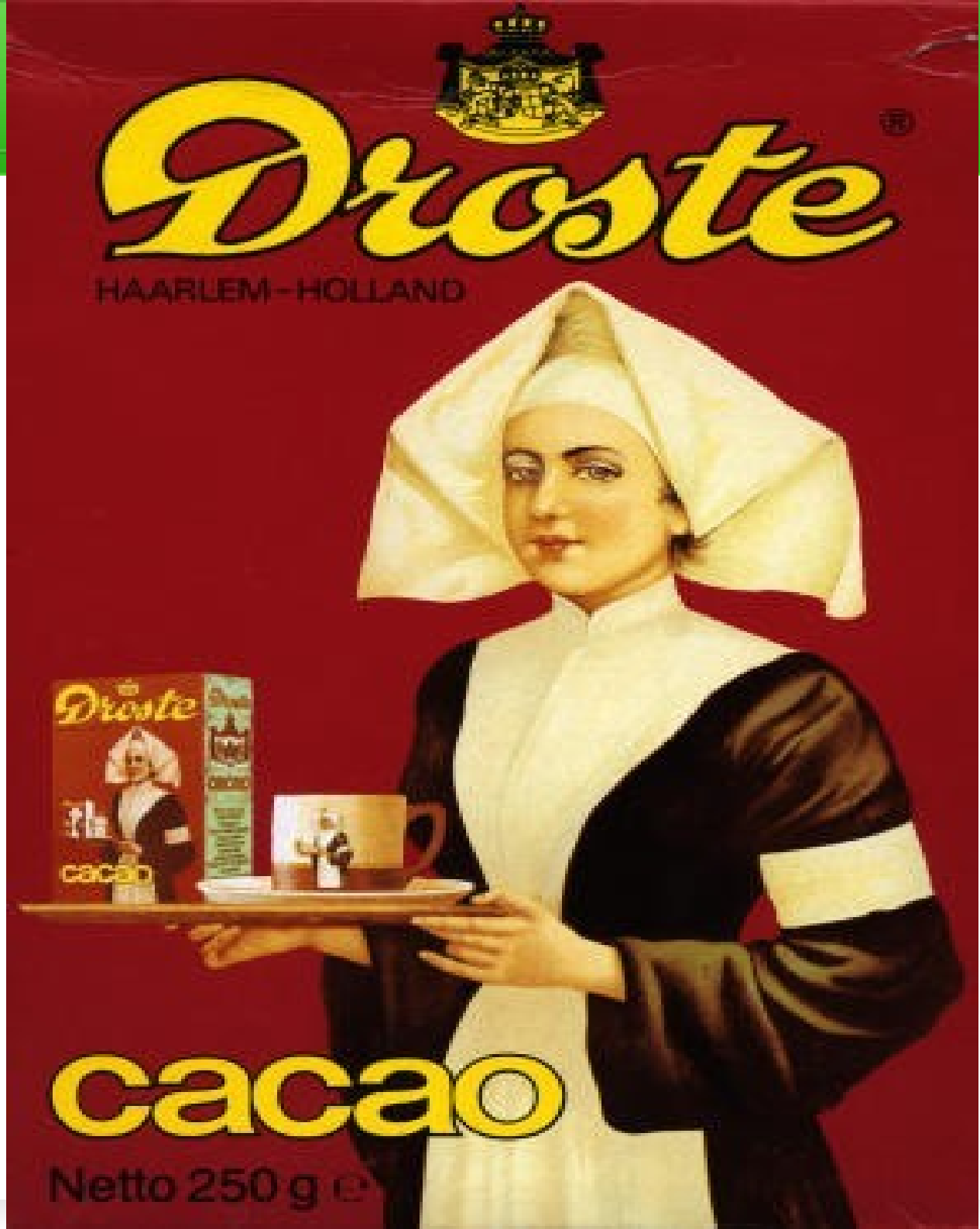
Recursividade



Recursividade é uma das coisas mágicas e interessantes em Programação.

Recursividade

Anuncio de cacao com uma imagem recursiva.



Recursividade

```
def contagem_regressiva(n):
    if n==0:
        print "Fogo!"
    else:
        print n
        contagem_regressiva(n-1)
```

Recursividade

Porque não usar Iteração ao invés de Recursividade?

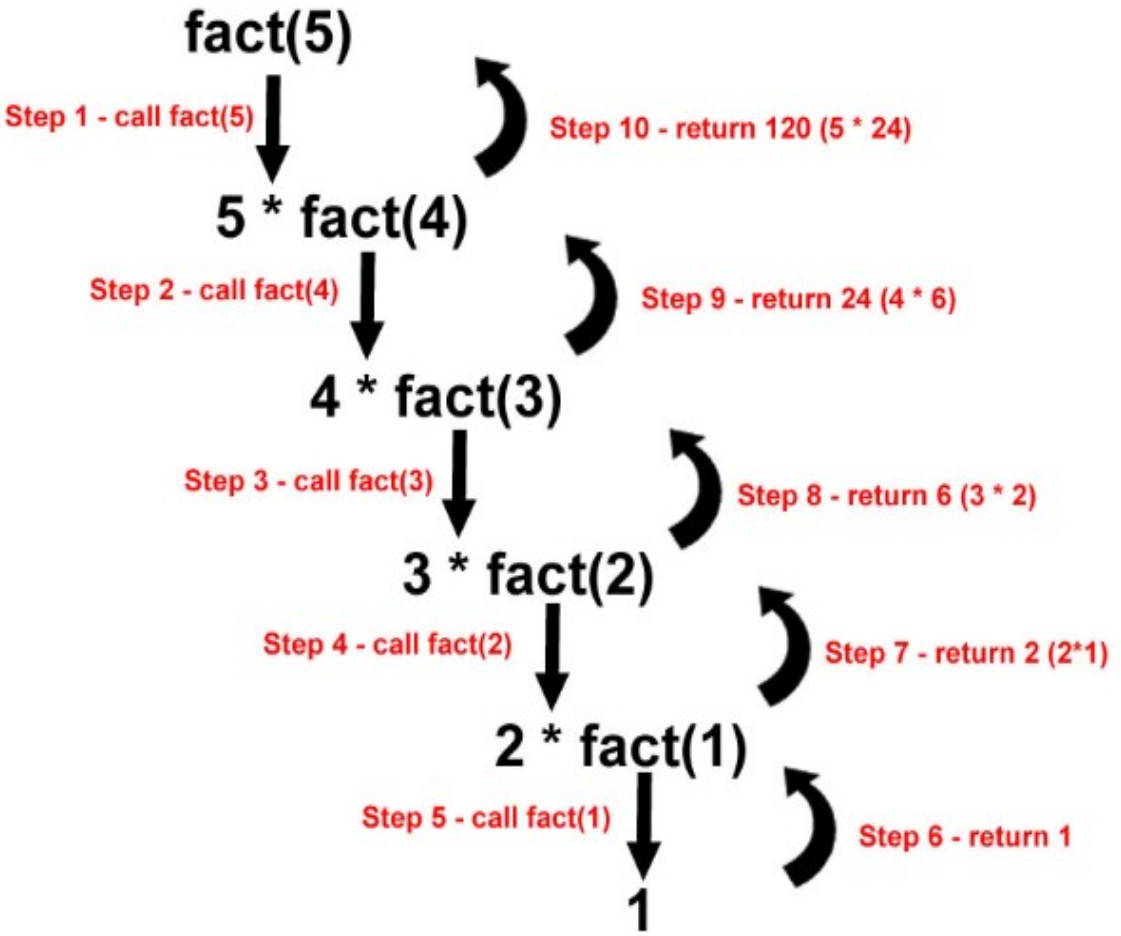
Depende muito do estilo de programação. Entretanto, algumas vezes é mais apropriado usar Recursividade para resolver um problema.

```
def contagem_regressiva(n):
    if n==0:
        print "Fogo!"
    else:
        print n
        contagem_regressiva(n-1)
```

```
def contagem_regressiva2(n):
    while n>0:
        print n
        n = n-1
    print "Fogo!"
```

Recursividade

```
def fact(n):
    if n==1:
        return 1
    else:
        return n*fact(n-1)
```



Busca em profundidade (versão recursiva)

```
DFS(G,s):
    Para cada vértice v em G.V-{s} faça
        v.cor = BRANCO
        v.t1  = INFINITO
        v.t2  = INFINITO
    tempo = 0
    VisitaDFS(G,s)
```

Inicialização

```
VisitaDFS(G,s):
    s.t1 = tempo
    s.cor = CINZA
    tempo = tempo+1
    Para cada v em G.Adj[s] faça
        Se v.cor == BRANCO
            VisitaDFS(G,v)
    s.cor = PRETO
    s.t2  = tempo
    tempo = tempo+1
```

Percorre o grafo

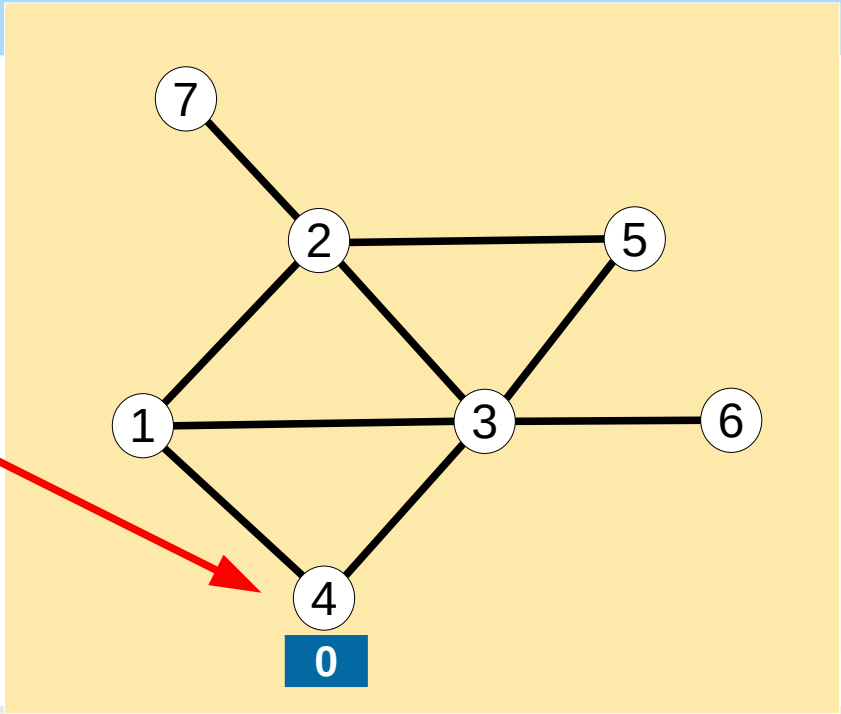
Busca em profundidade (versão recursiva)

```
VisitaDFS(G,s):
s.t1 = tempo
s.cor = CINZA
tempo = tempo+1
Para cada v em G.Adj[s] faça
    Se v.cor == BRANCO
        VisitaDFS(G,v)
s.cor = PRETO
s.t2  = tempo
tempo = tempo+1
```

Percorre o grafo

s=4

tempo=0



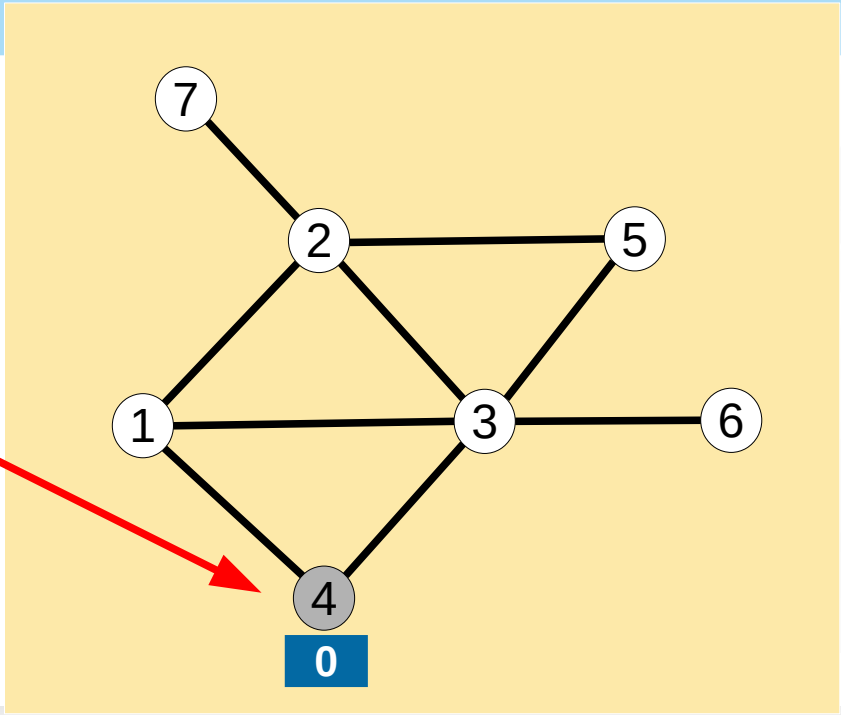
Busca em profundidade (versão recursiva)

```
VisitaDFS(G,s):
   s.t1 = tempo
   s.cor = CINZA
   tempo = tempo+1
   Para cada v em G.Adj[s] faça
       Se v.cor == BRANCO
           VisitaDFS(G,v)
   s.cor = PRETO
   s.t2  = tempo
   tempo = tempo+1
```

Percorre o grafo

s=4

tempo=0



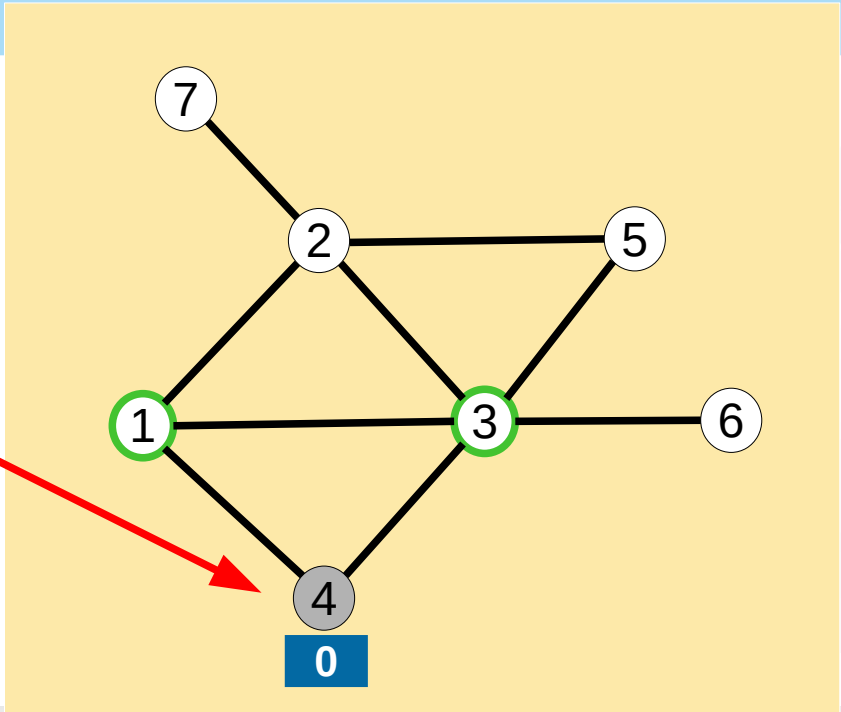
Busca em profundidade (versão recursiva)

```
VisitaDFS(G,s):
  s.t1 = tempo
  s.cor = CINZA
  tempo = tempo+1
  Para cada v em G.Adj[s] faça
      Se v.cor == BRANCO
          VisitaDFS(G,v)
  s.cor = PRETO
  s.t2  = tempo
  tempo = tempo+1
```

Percorre o grafo

s=4

tempo=1



Busca em profundidade (versão recursiva)

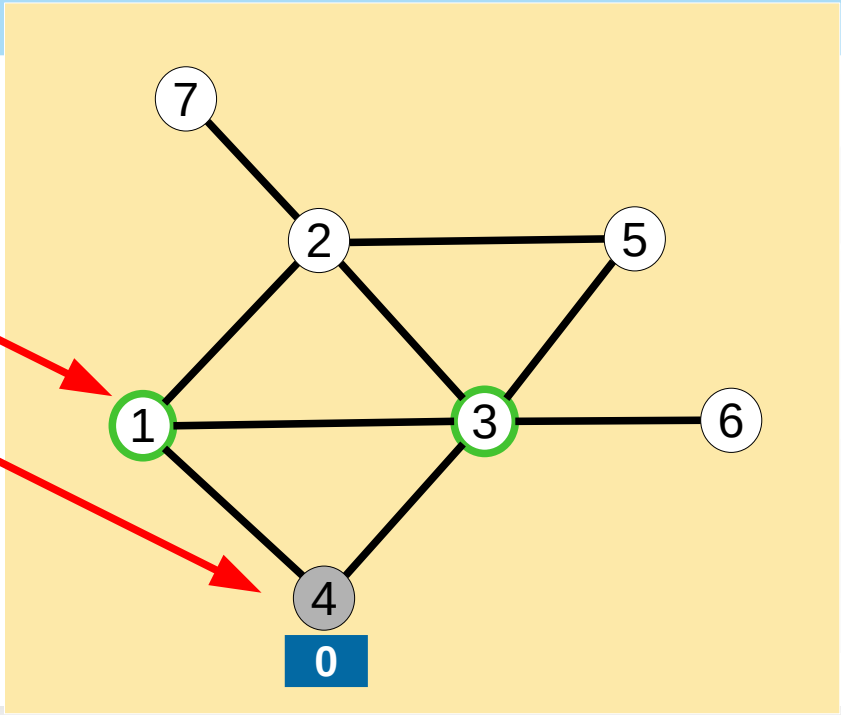
```
VisitaDFS(G,s):
  s.t1 = tempo
  s.cor = CINZA
  tempo = tempo+1
  Para cada v em G.Adj[s] faça
      Se v.cor == BRANCO
          VisitaDFS(G,v)
  s.cor = PRETO
  s.t2  = tempo
  tempo = tempo+1
```

Percorre o grafo

v=1

s=4

tempo=2



Busca em profundidade (versão recursiva)

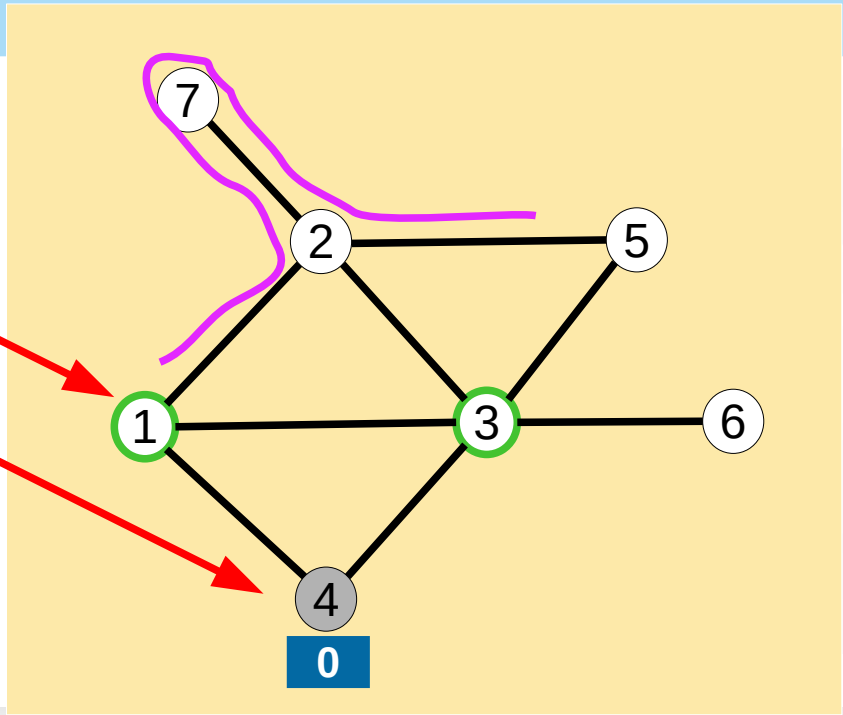
```
VisitaDFS(G,s):
  s.t1 = tempo
  s.cor = CINZA
  tempo = tempo+1
  Para cada v em G.Adj[s] faça
      Se v.cor == BRANCO
          VisitaDFS(G,v)
  s.cor = PRETO
  s.t2  = tempo
  tempo = tempo+1
```

Percorre o grafo

v=1

s=4

tempo=3



Busca em profundidade (versão recursiva)

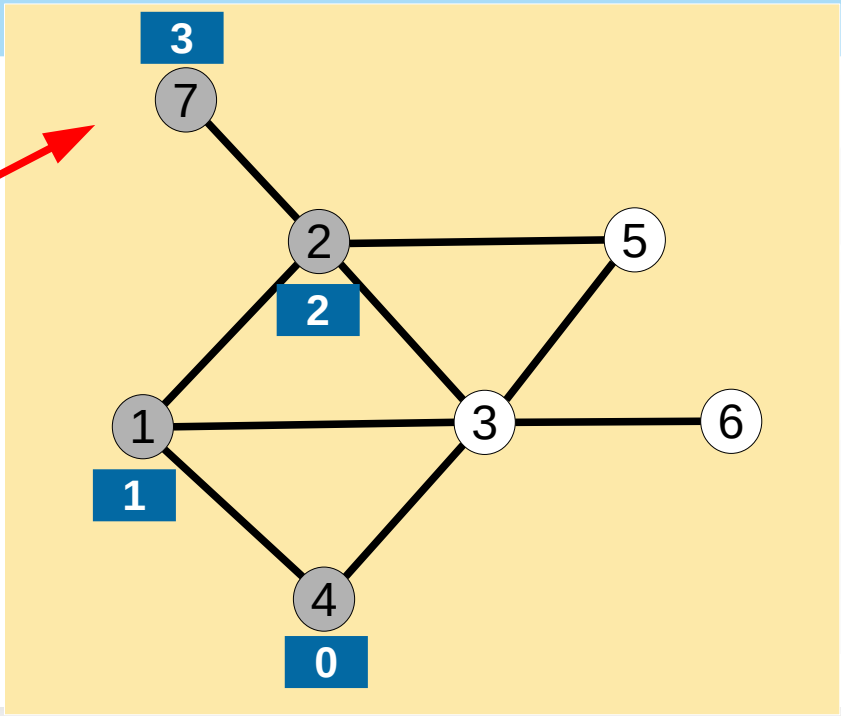
VisitaDFS(G,s):

```
s.t1 = tempo
s.cor = CINZA
tempo = tempo+1
Para cada v em G.Adj[s] faça
    Se v.cor == BRANCO
        VisitaDFS(G,v)
s.cor = PRETO
s.t2  = tempo
tempo = tempo+1
```

Percorre o grafo

s=7

tempo=4



Após alguns chamados recursivos...

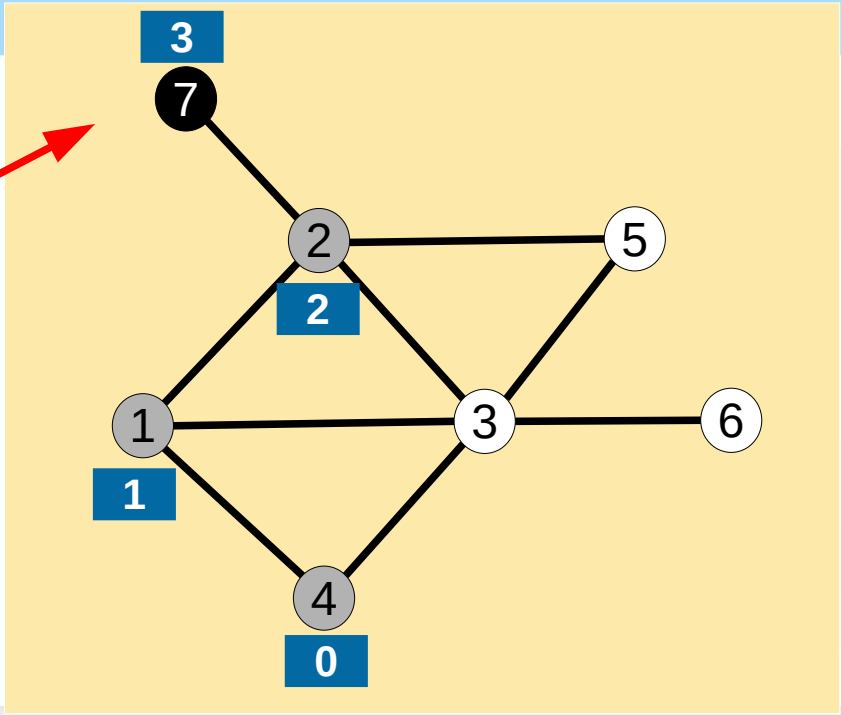
Busca em profundidade (versão recursiva)

```
VisitaDFS(G,s):
    s.t1 = tempo
    s.cor = CINZA
    tempo = tempo+1
    Para cada v em G.Adj[s] faça
        Se v.cor == BRANCO
            VisitaDFS(G,v)
    s.cor = PRETO
    s.t2  = tempo
    tempo = tempo+1
```

Percorre o grafo

s=7

tempo=4



Após alguns chamados recursivos...

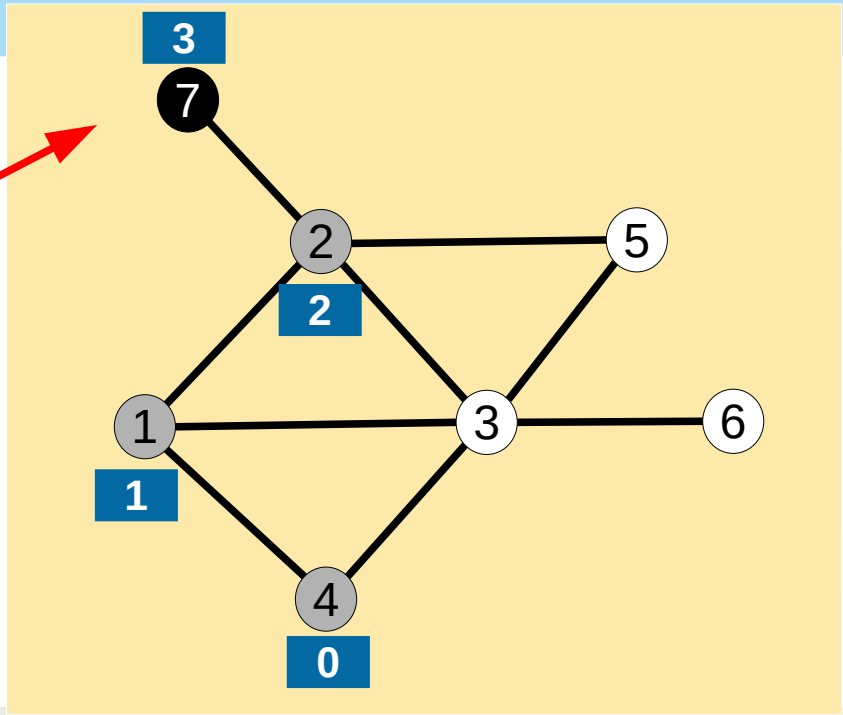
Busca em profundidade (versão recursiva)

```
VisitaDFS(G,s):
    s.t1 = tempo
    s.cor = CINZA
    tempo = tempo+1
    Para cada v em G.Adj[s] faça
        Se v.cor == BRANCO
            VisitaDFS(G,v)
    s.cor = PRETO
    s.t2  = tempo
    tempo = tempo+1
```

Percorre o grafo

s=7

tempo=4



Após alguns chamados recursivos...

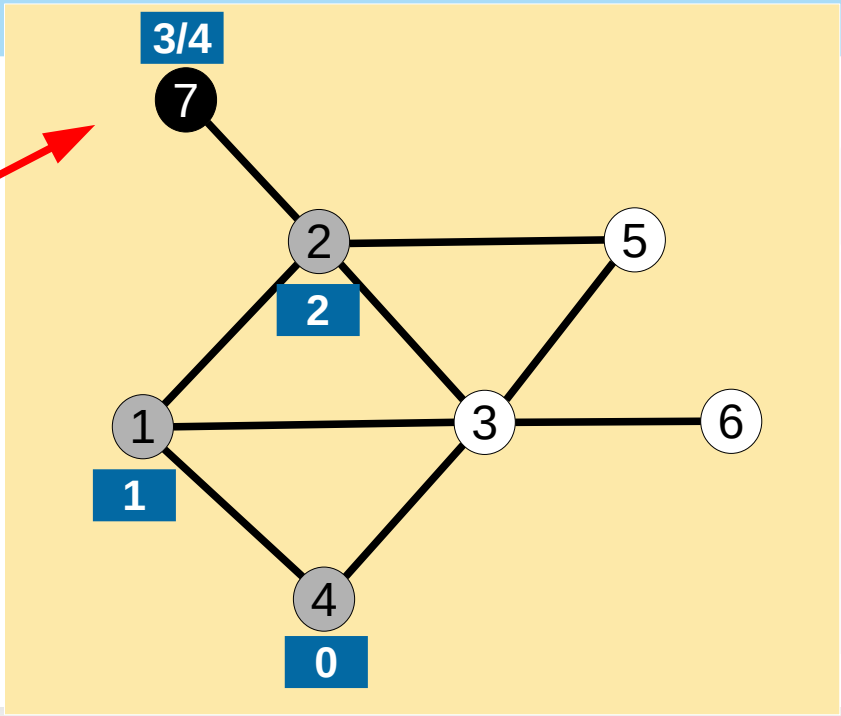
Busca em profundidade (versão recursiva)

```
VisitaDFS(G,s):
    s.t1 = tempo
    s.cor = CINZA
    tempo = tempo+1
    Para cada v em G.Adj[s] faça
        Se v.cor == BRANCO
            VisitaDFS(G,v)
    s.cor = PRETO
    s.t2  = tempo
    tempo = tempo+1
```

Percorre o grafo

s=7

tempo=5



Após alguns chamados recursivos...

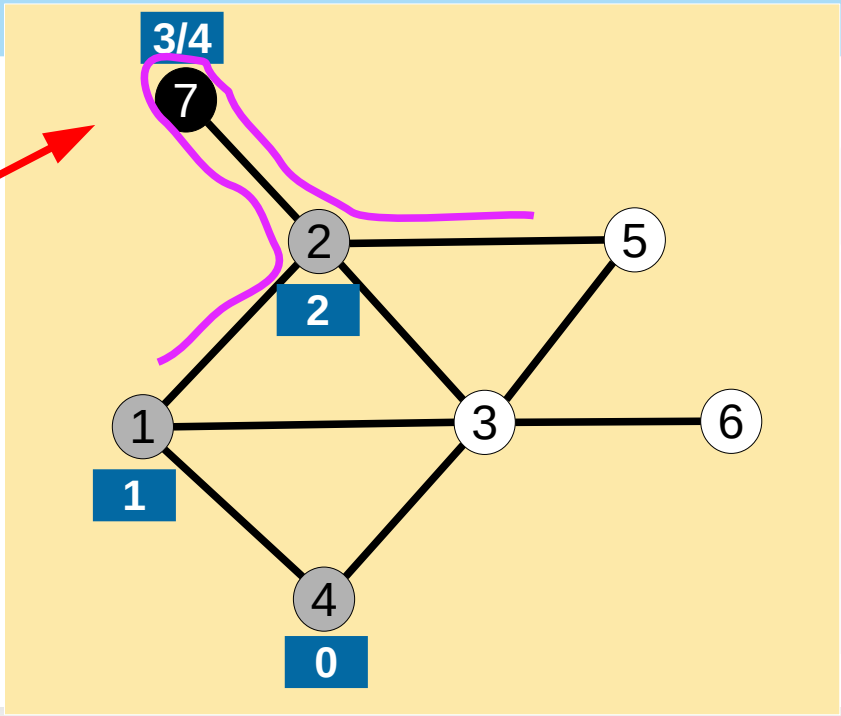
Busca em profundidade (versão recursiva)

```
VisitaDFS(G,s):
    s.t1 = tempo
    s.cor = CINZA
    tempo = tempo+1
    Para cada v em G.Adj[s] faça
        Se v.cor == BRANCO
            VisitaDFS(G,v)
    s.cor = PRETO
    s.t2  = tempo
    tempo = tempo+1
```

Percorre o grafo

s=7

tempo=5



Após alguns chamados recursivos...

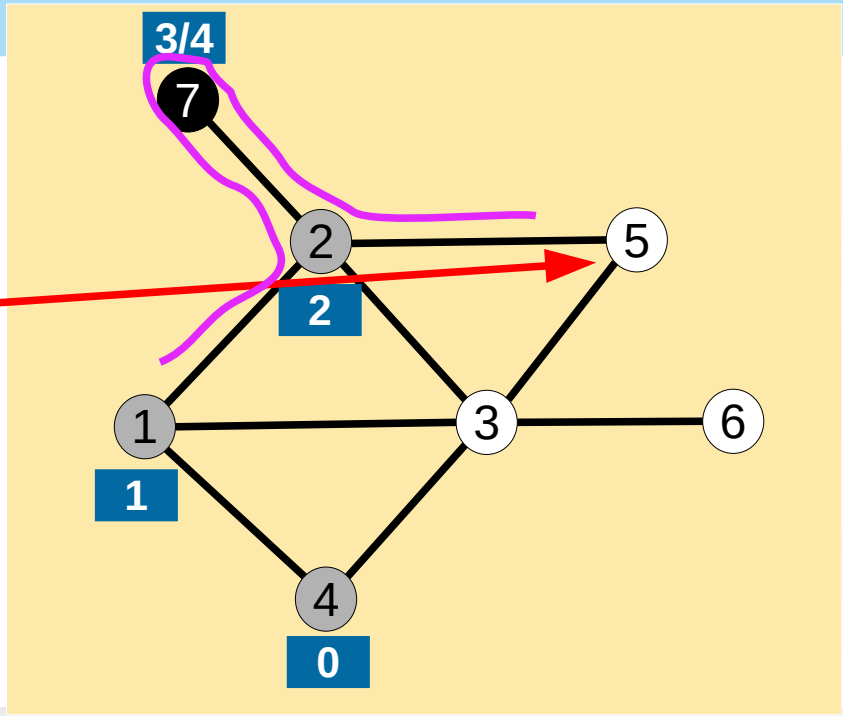
Busca em profundidade (versão recursiva)

```
VisitaDFS(G,s):
    s.t1 = tempo
    s.cor = CINZA
    tempo = tempo+1
    Para cada v em G.Adj[s] faça
        Se v.cor == BRANCO
            VisitaDFS(G,v)
    s.cor = PRETO
    s.t2  = tempo
    tempo = tempo+1
```

Percorre o grafo

s=5

tempo=5



Após alguns chamados recursivos...

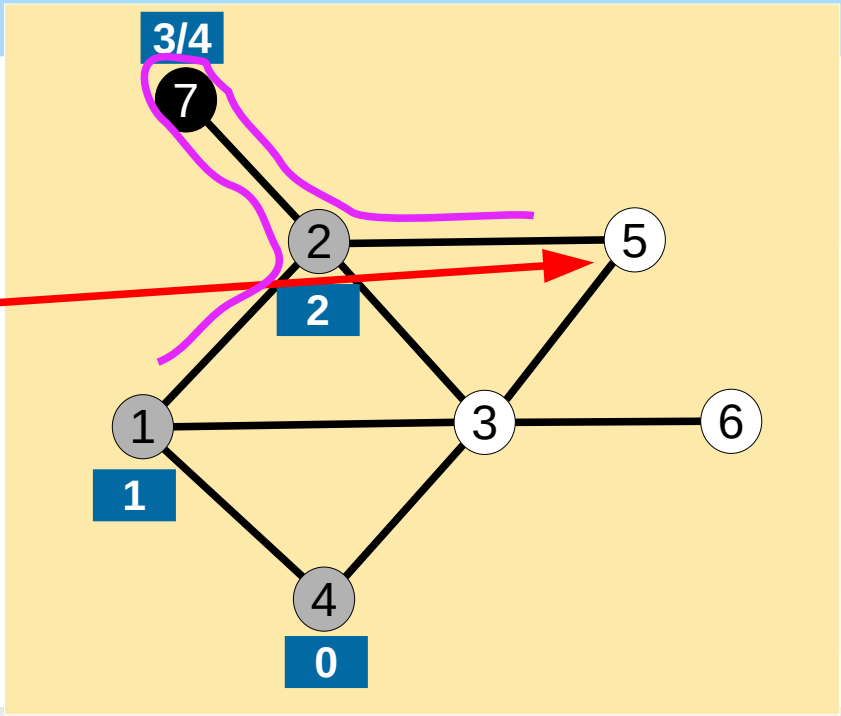
Busca em profundidade (versão recursiva)

```
VisitaDFS(G,s):
s.t1 = tempo
s.cor = CINZA
tempo = tempo+1
Para cada v em G.Adj[s] faça
    Se v.cor == BRANCO
        VisitaDFS(G,v)
s.cor = PRETO
s.t2  = tempo
tempo = tempo+1
```

Percorre o grafo

s=5

tempo=5



Após alguns chamados recursivos...

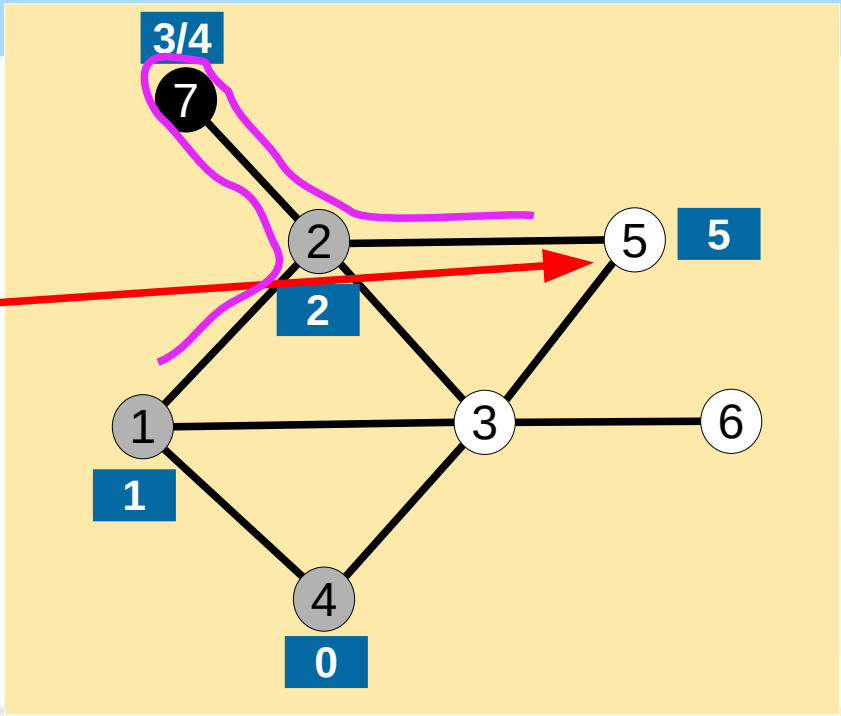
Busca em profundidade (versão recursiva)

```
VisitaDFS(G,s):
    s.t1 = tempo
    s.cor = CINZA
    tempo = tempo+1
    Para cada v em G.Adj[s] faça
        Se v.cor == BRANCO
            VisitaDFS(G,v)
    s.cor = PRETO
    s.t2  = tempo
    tempo = tempo+1
```

Percorre o grafo

s=5

tempo=5



Após alguns chamados recursivos...

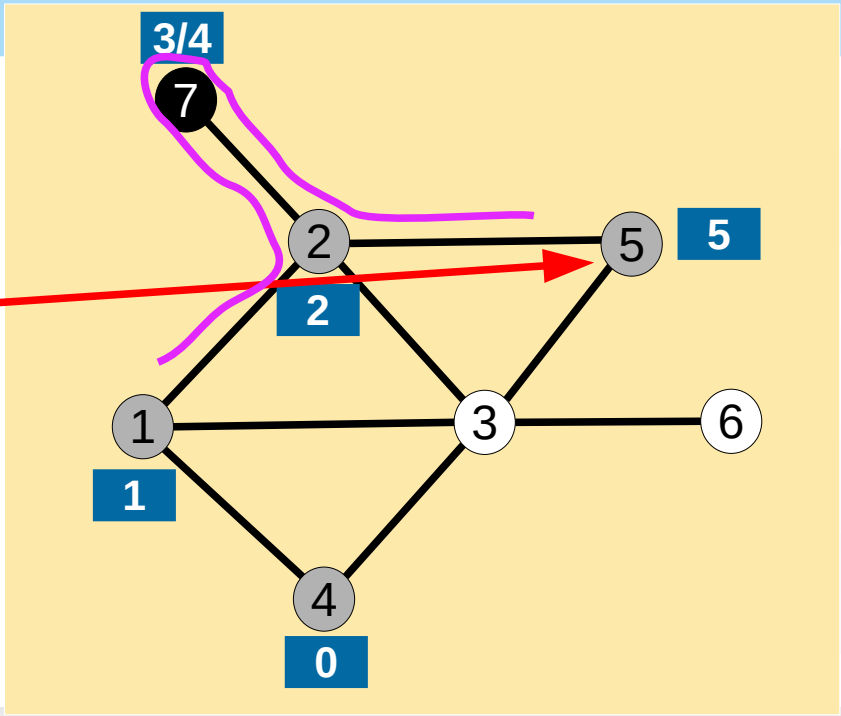
Busca em profundidade (versão recursiva)

```
VisitaDFS(G,s):
  s.t1 = tempo
  s.cor = CINZA
  tempo = tempo+1
  Para cada v em G.Adj[s] faça
      Se v.cor == BRANCO
          VisitaDFS(G,v)
  s.cor = PRETO
  s.t2  = tempo
  tempo = tempo+1
```

Percorre o grafo

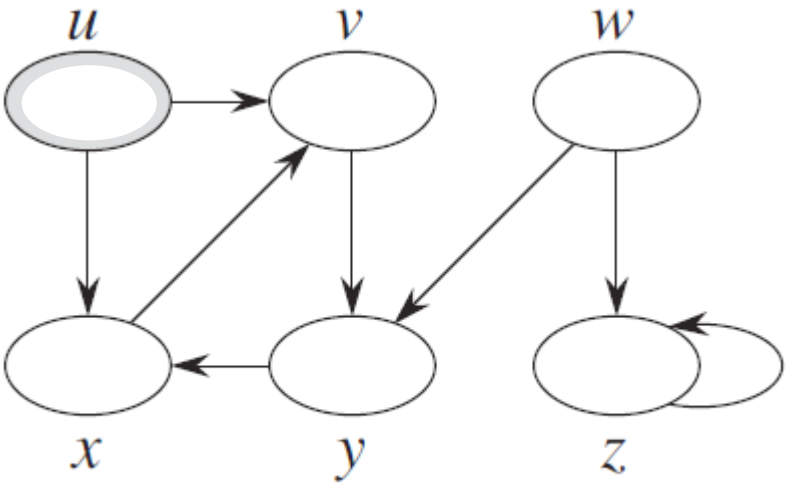
s=5

tempo=6

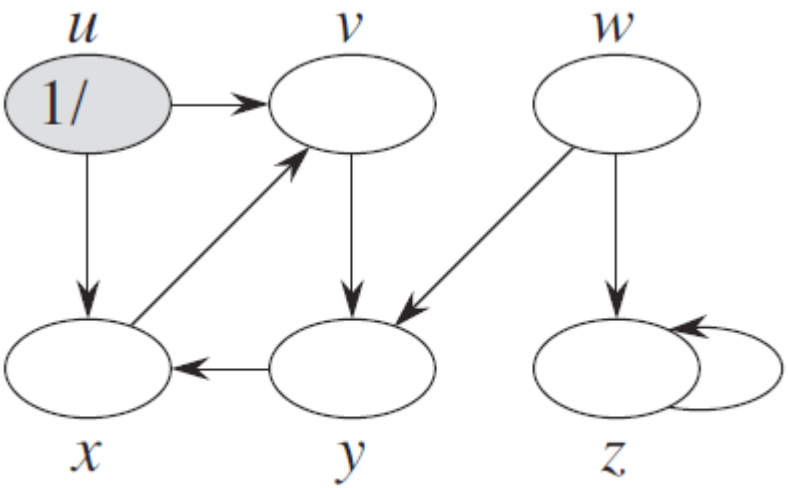


Após alguns chamados recursivos...

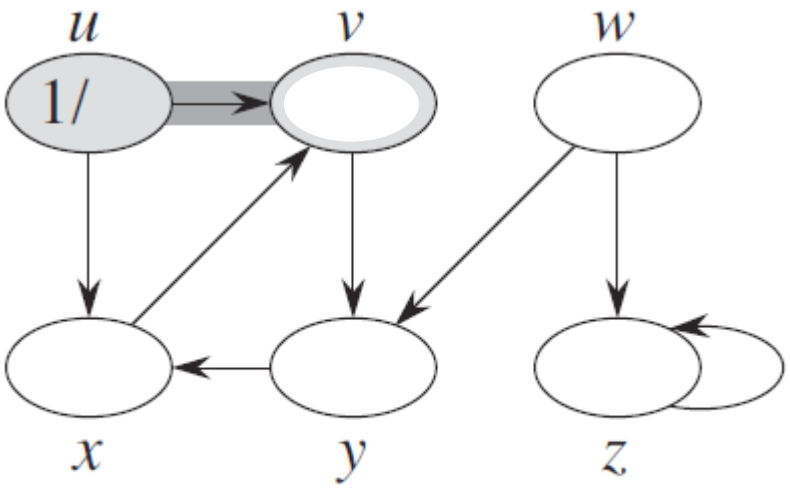
Para grafos direcionados



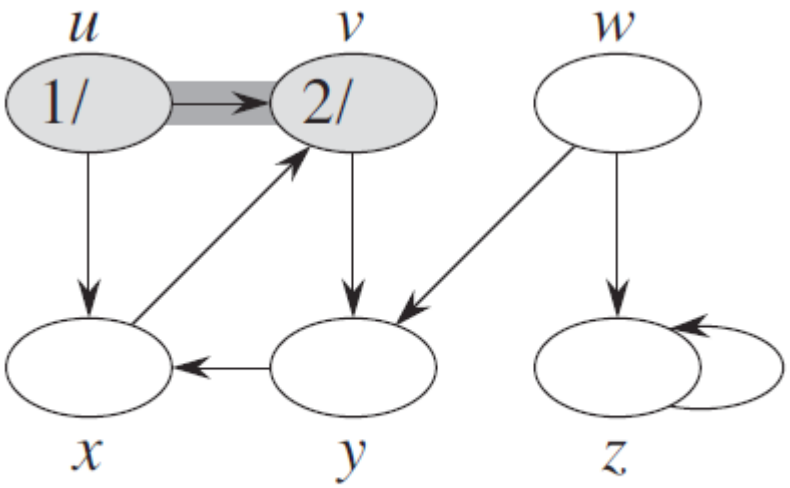
Para grafos direcionados



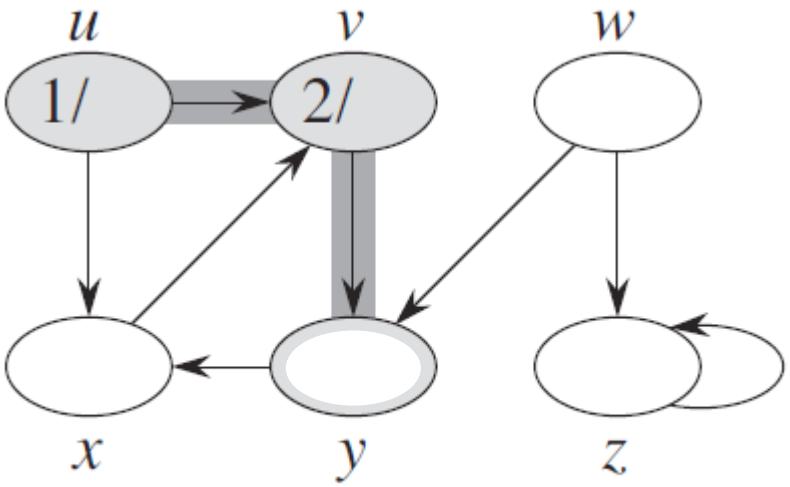
Para grafos direcionados



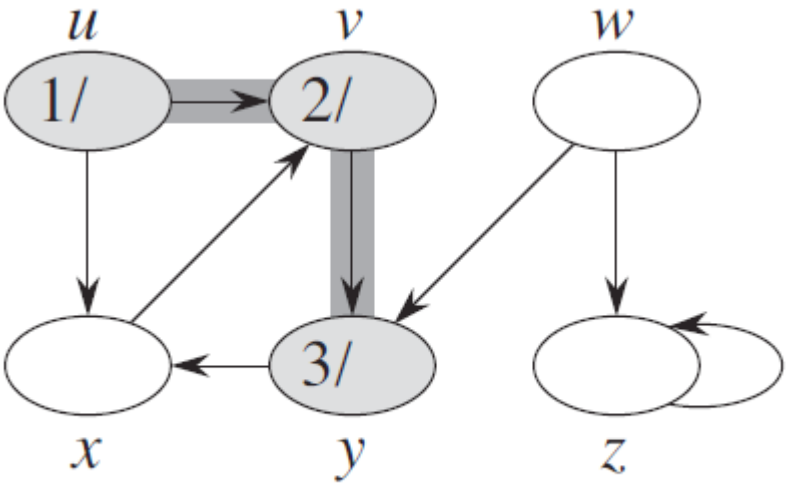
Para grafos direcionados



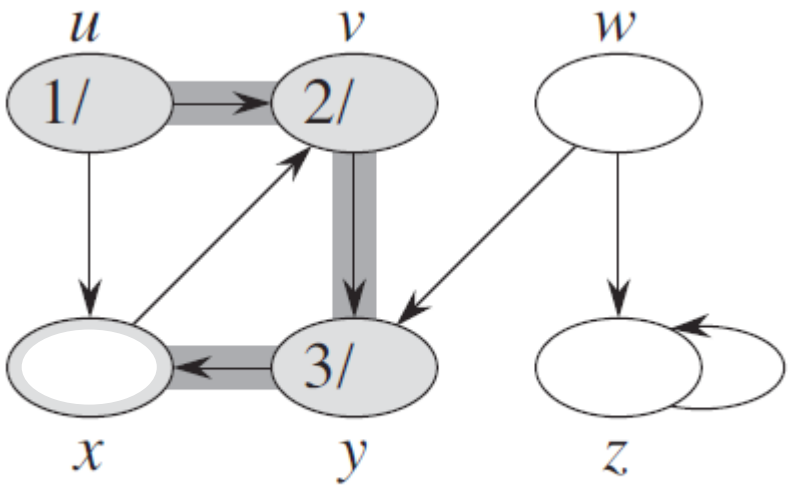
Para grafos direcionados



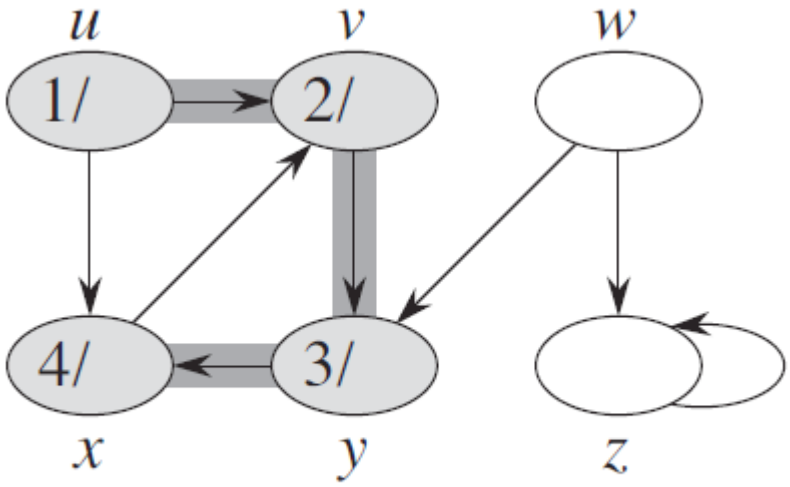
Para grafos direcionados



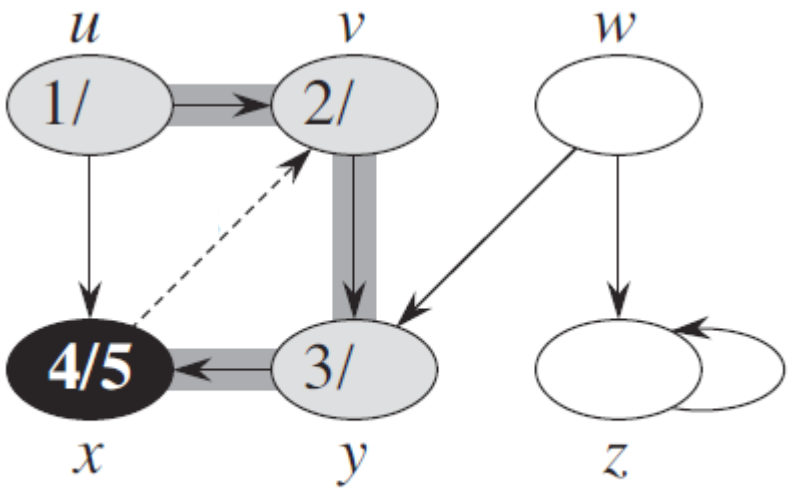
Para grafos direcionados



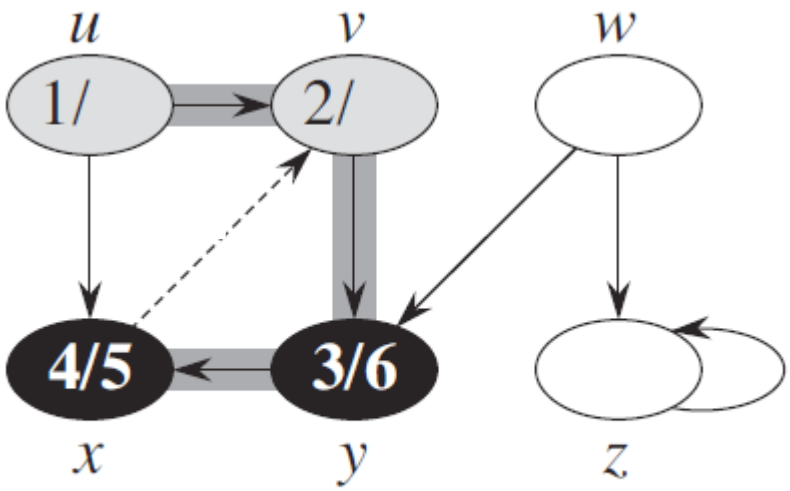
Para grafos direcionados



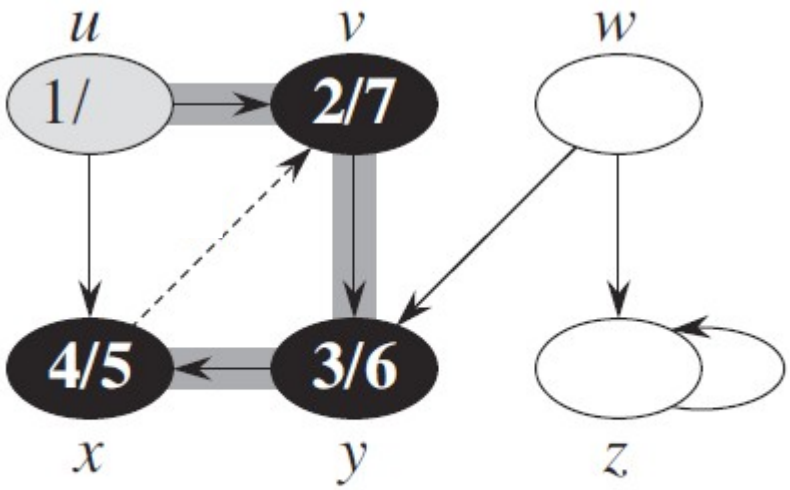
Para grafos direcionados



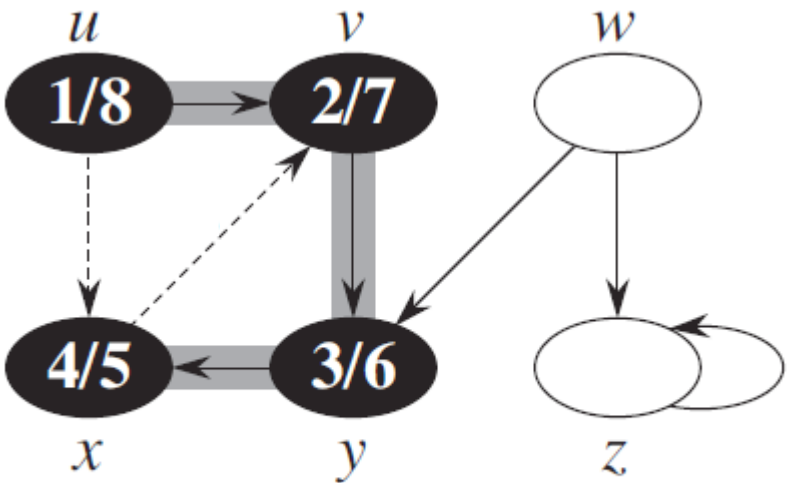
Para grafos direcionados



Para grafos direcionados



Para grafos direcionados

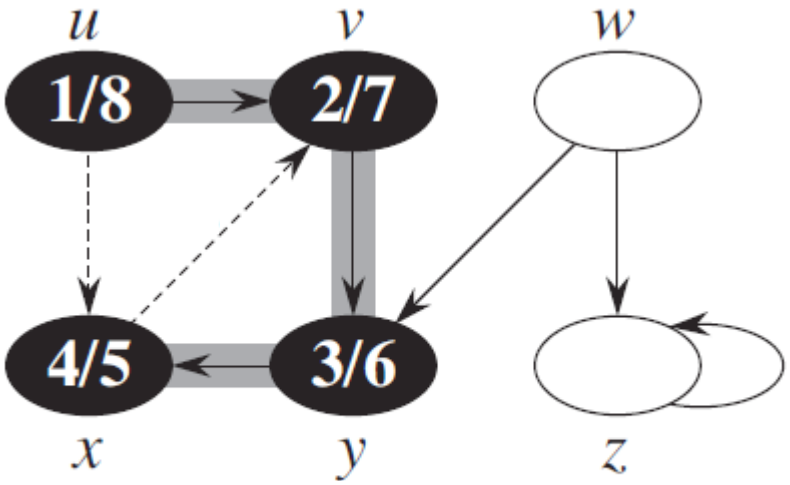


Busca em profundidade

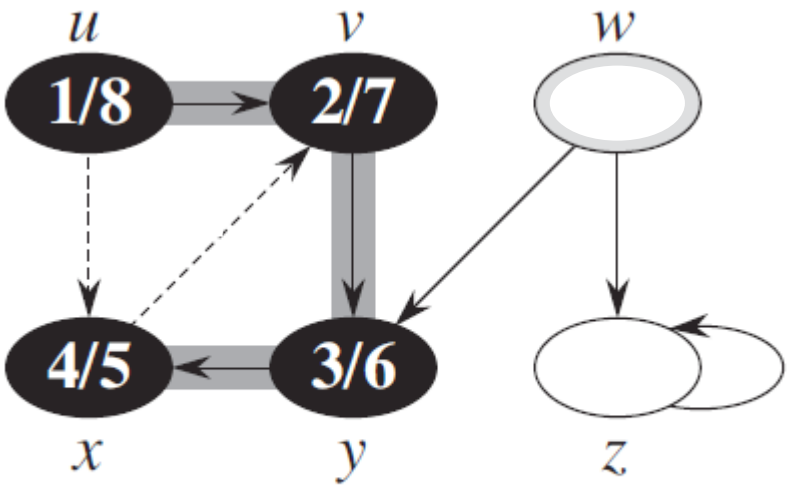
```
// Esta função permite percorrer os elementos da componente conexa contendo s.
DFS(G,s):
    Para cada vértice v em G.V-{s} faça                    Inicialização
        v.cor = BRANCO
        v.t1  = INFINITO
        v.t2  = INFINITO
    tempo = 0
    VisitaDFS(G,s)

// Esta nova função permite percorrer todos os elementos do grafo
DFS(G):
    Para cada vértice v em G.V faça                        Inicialização
        v.cor = BRANCO
        v.t1  = INFINITO
        v.t2  = INFINITO
    tempo = 0
    Para cada vértice u em G.V faça
        se u.cor==BRANCO
            VisitaDFS(G,u)
```

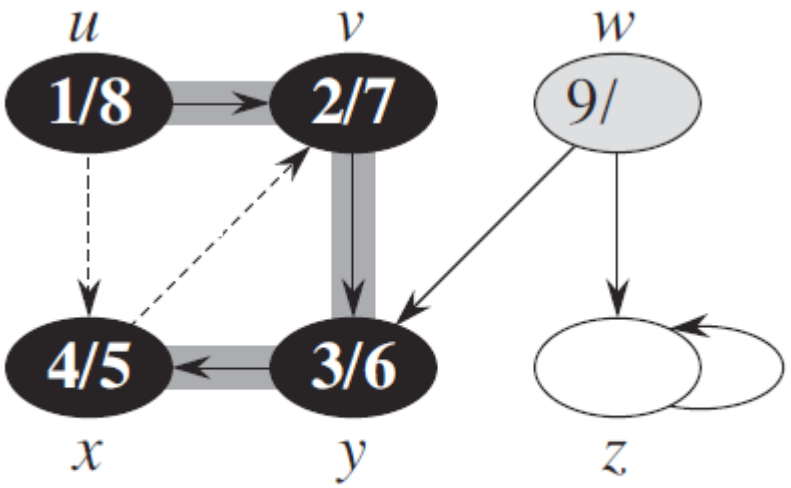
Para grafos direcionados



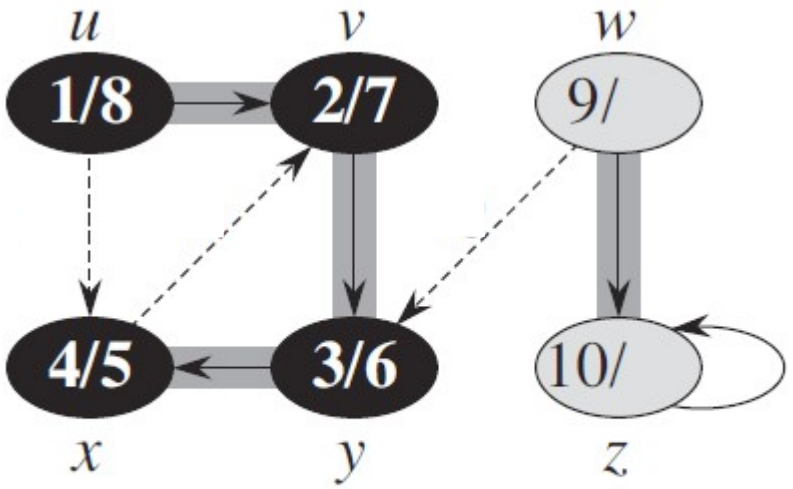
Para grafos direcionados



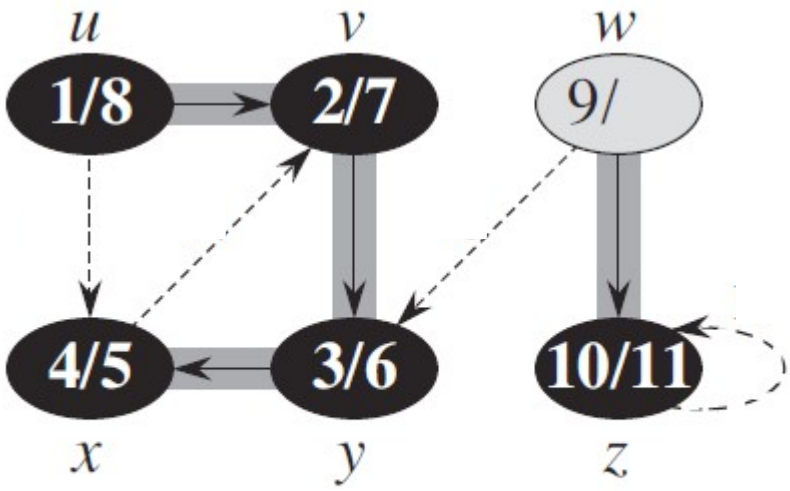
Para grafos direcionados



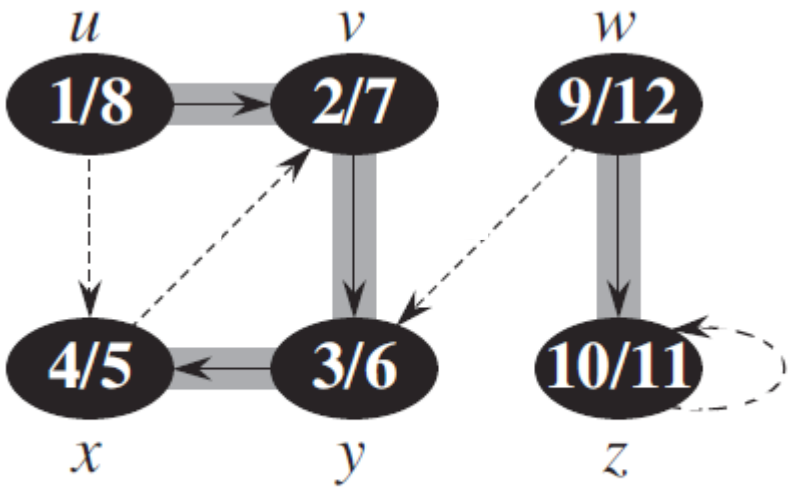
Para grafos direcionados



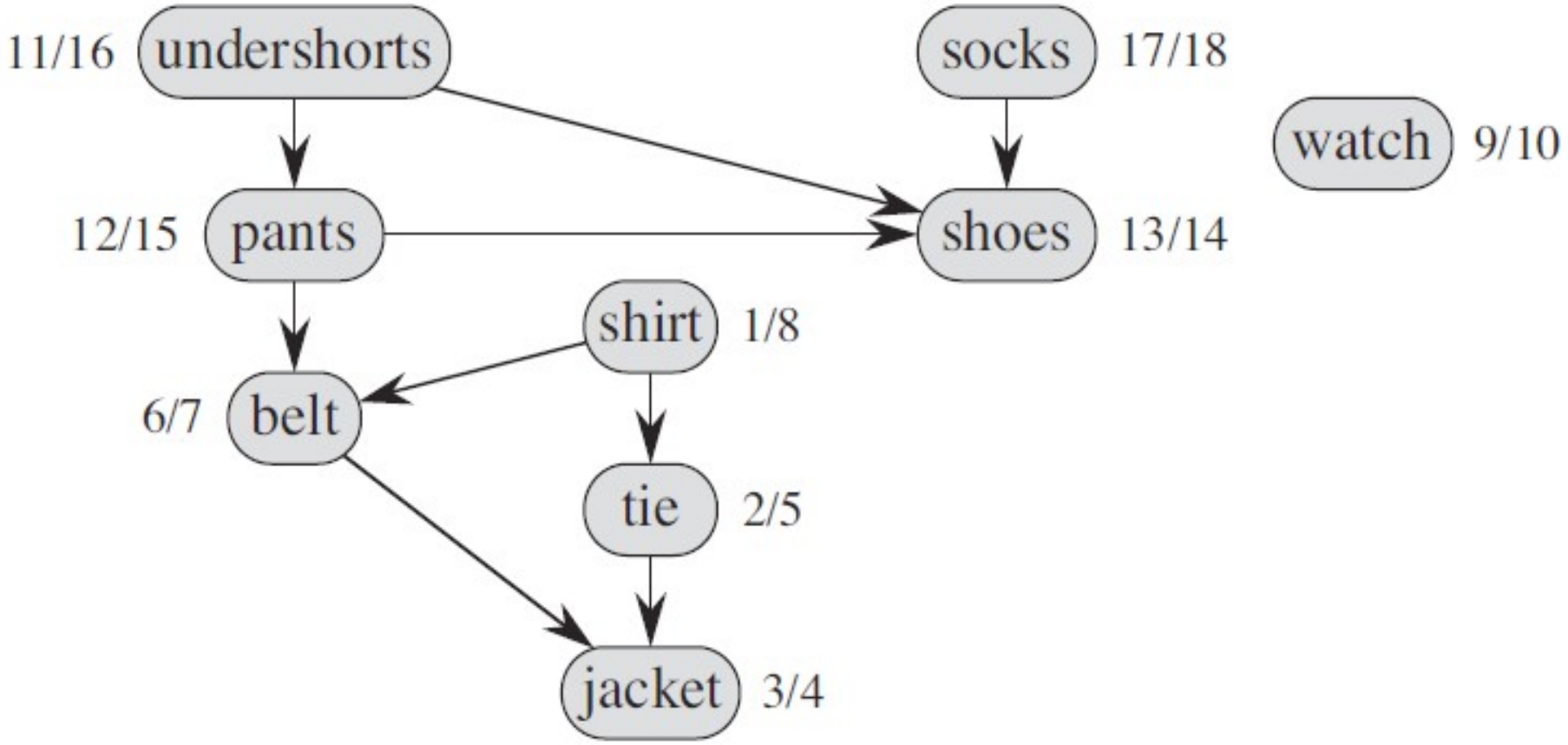
Para grafos direcionados



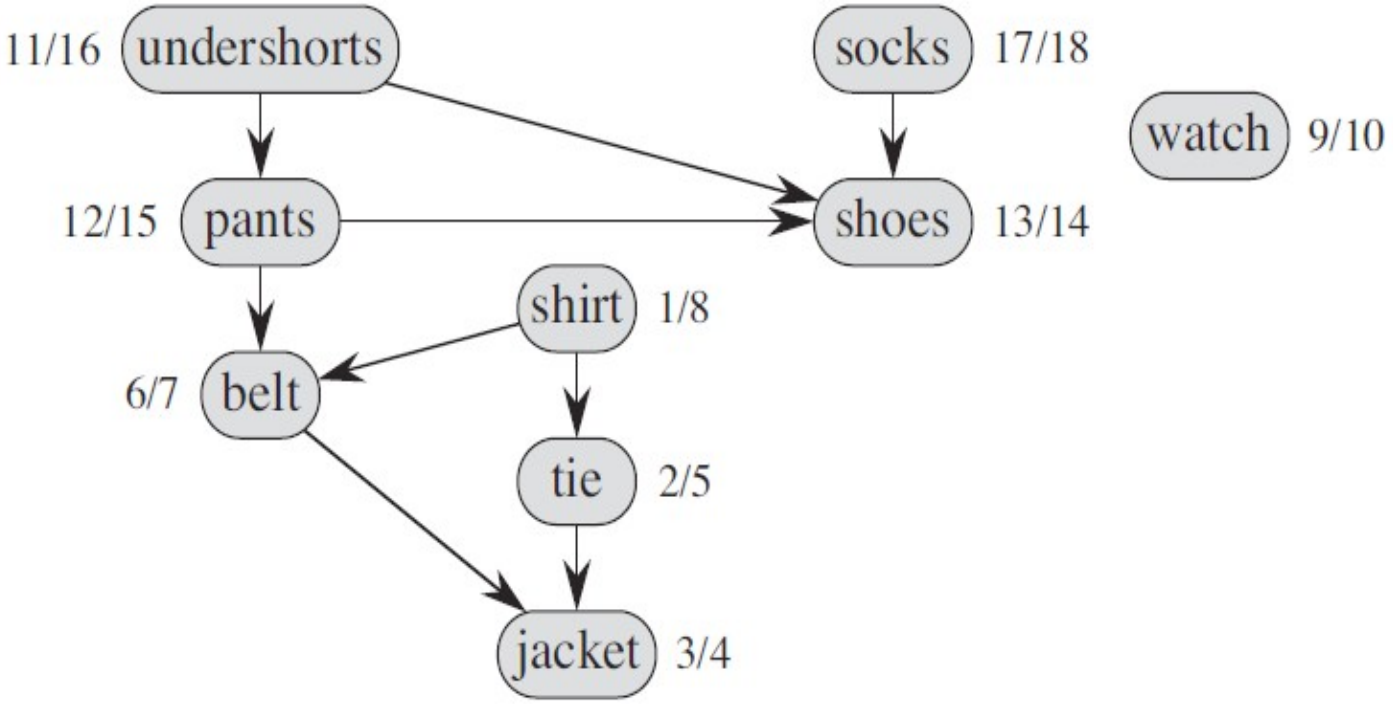
Para grafos direcionados

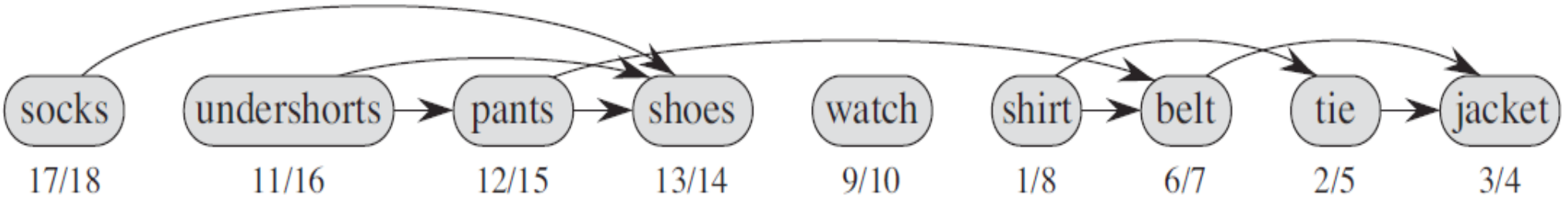


Ordenação 'topologica'



Ordenação 'topologica'



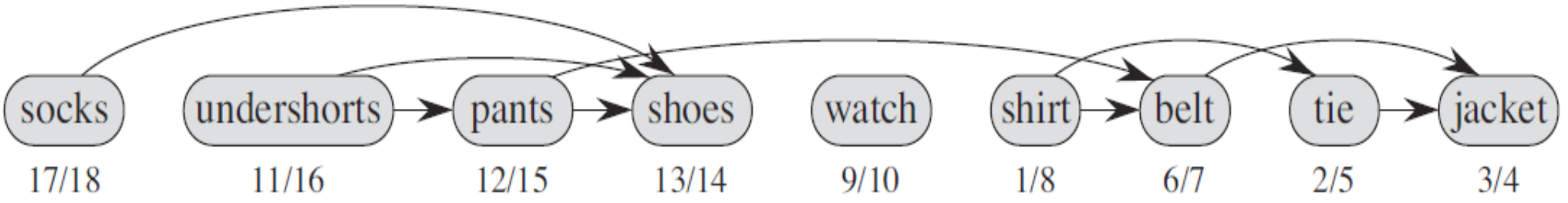


Ordenação 'topologica'

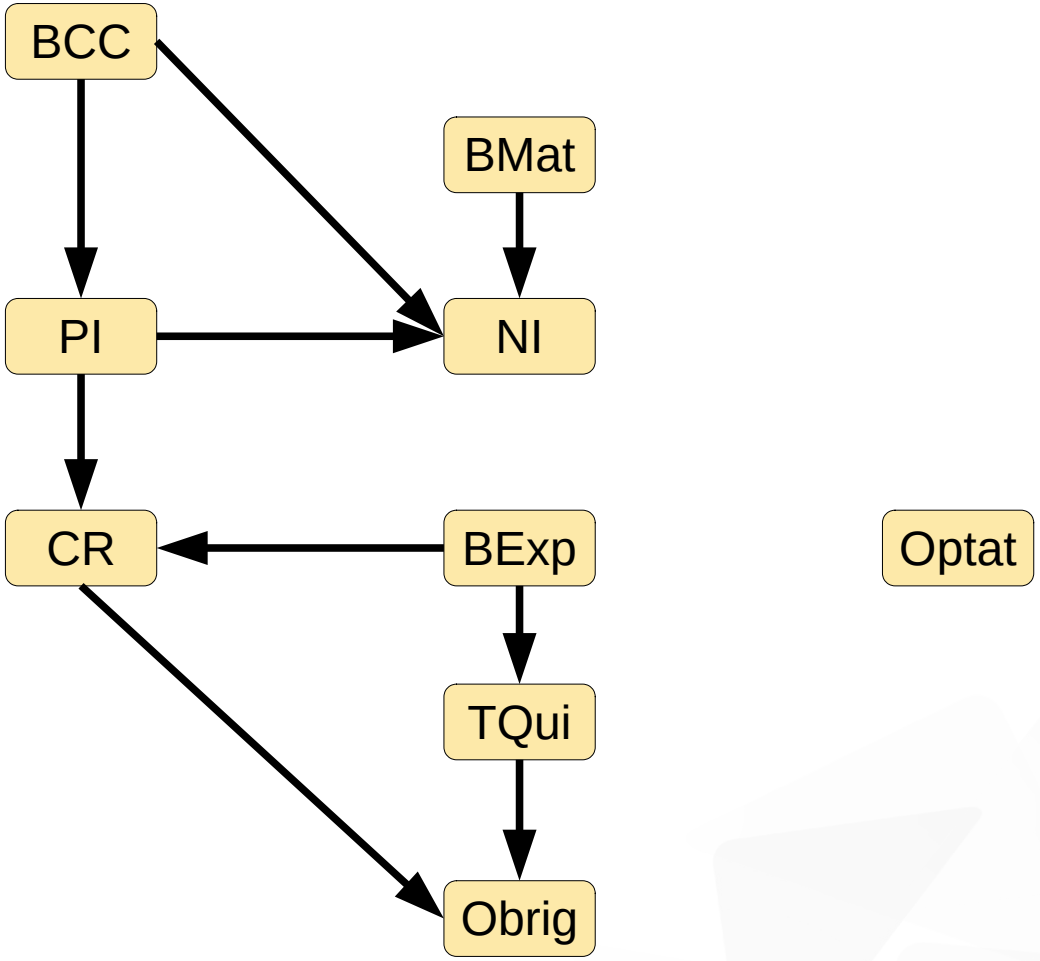
A **ordenação topologica (de um grafo direcionado)** é uma ordem linear de seus vértices em que:

- Cada aresta direcionada **uv** (do vértice **u** ao vértice **v**), o vértice **u** vem antes do vértice **v** na ordenação.

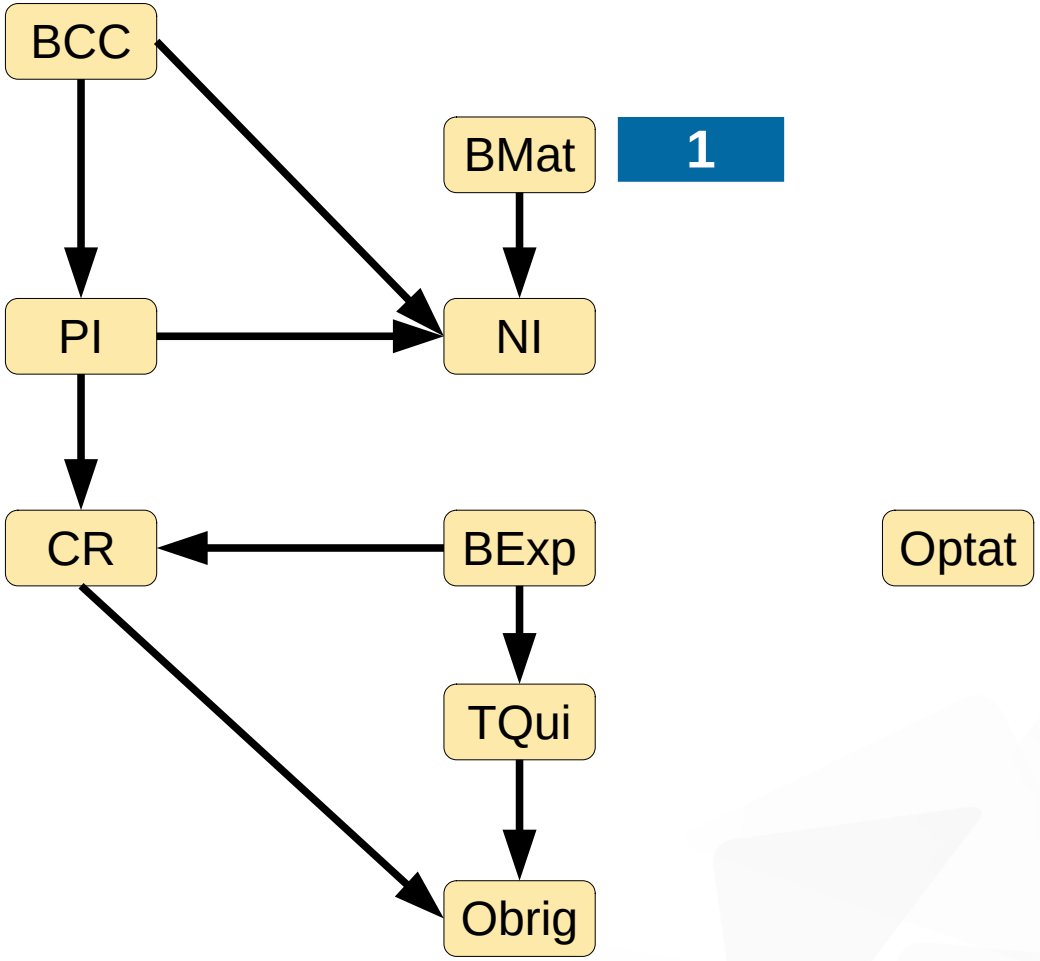
Podem existir uma ou mais ordenações topológicas.



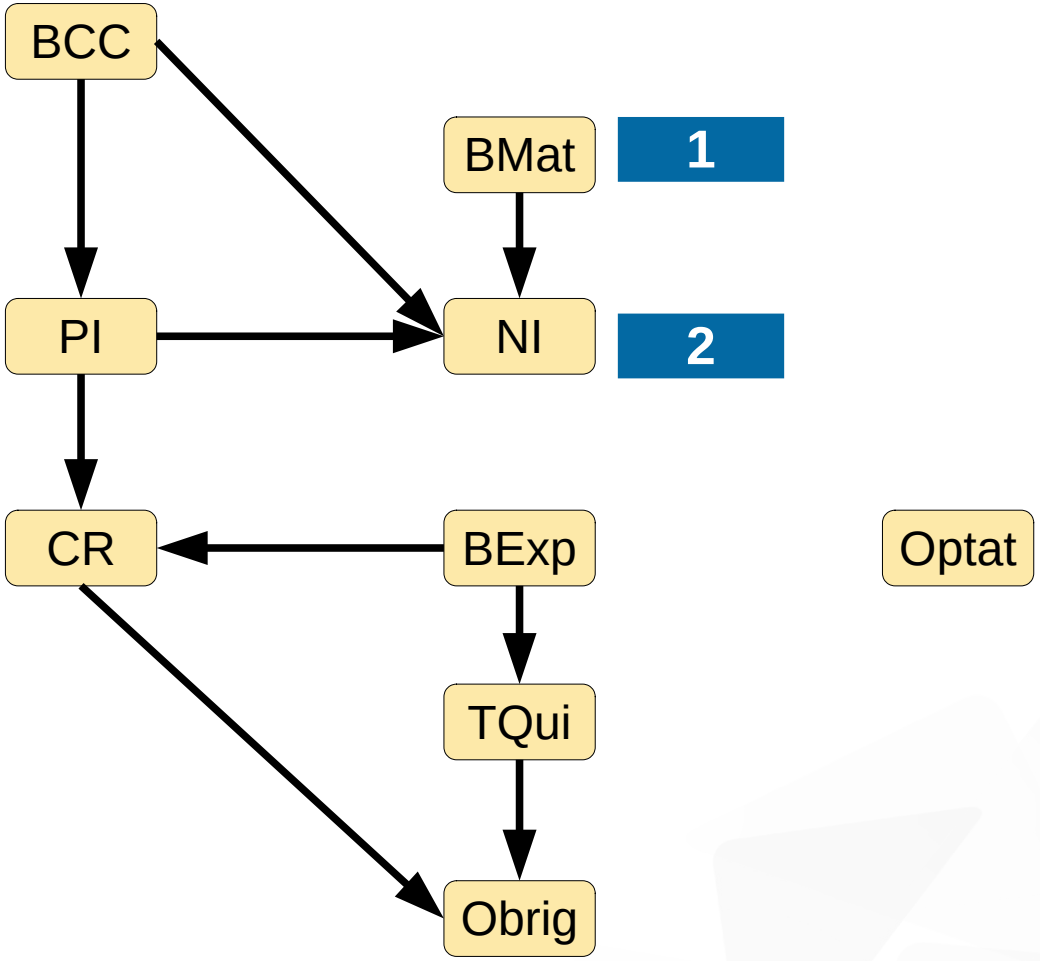
Ordenação 'topologica'



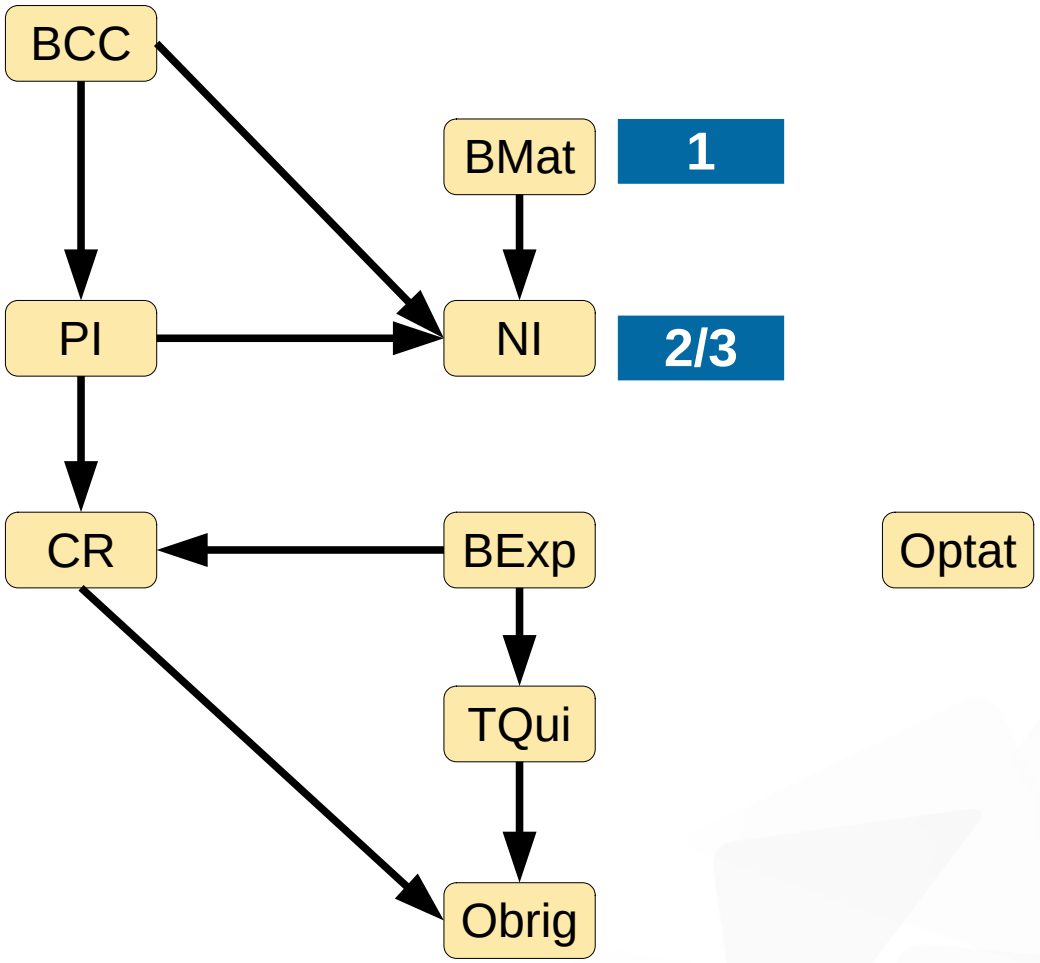
Ordenação 'topologica'



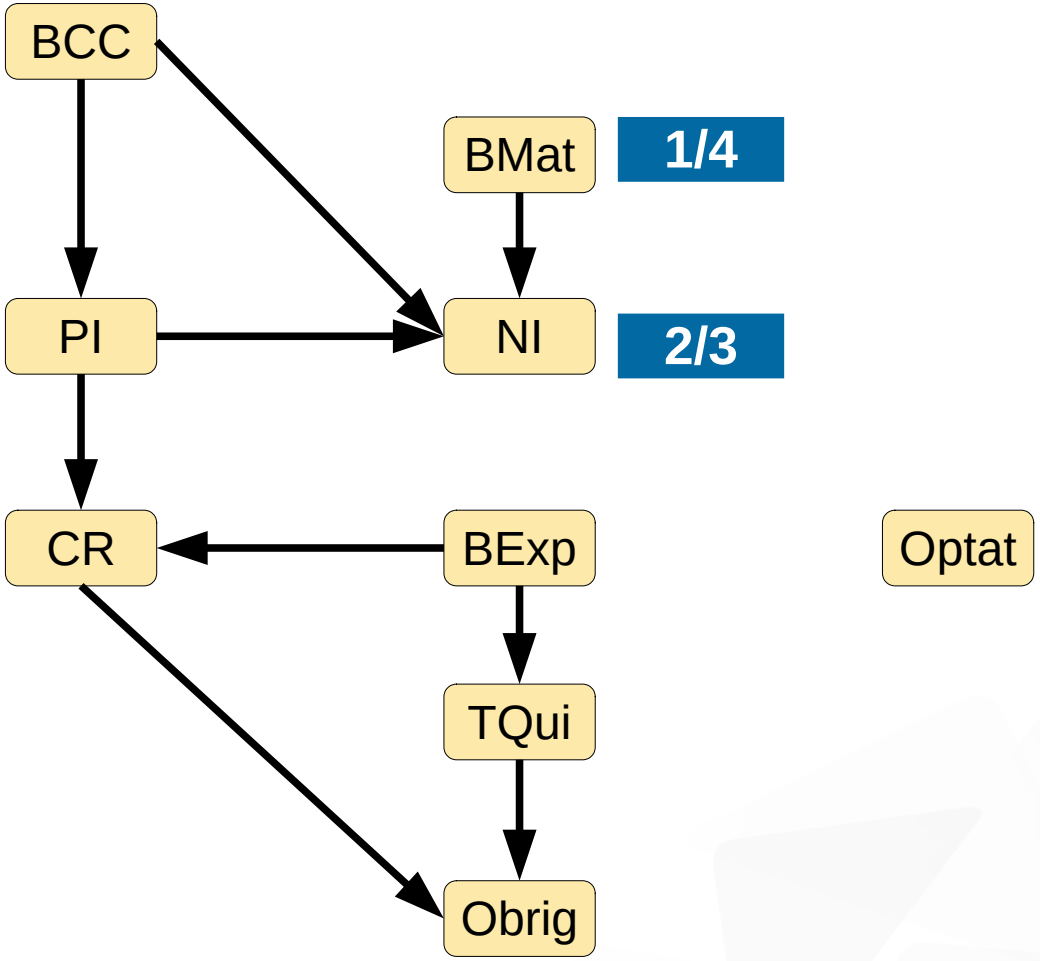
Ordenação 'topologica'



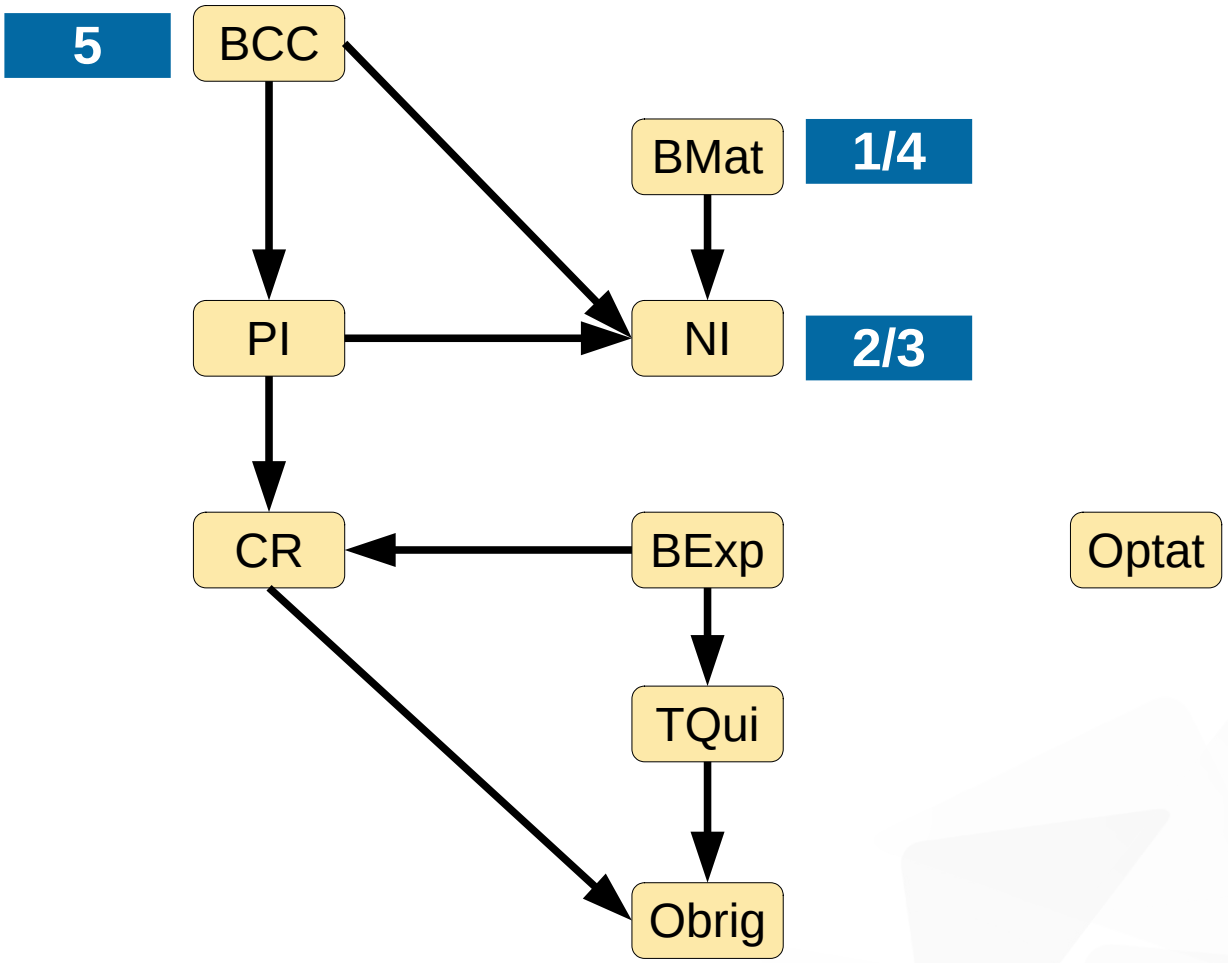
Ordenação 'topologica'



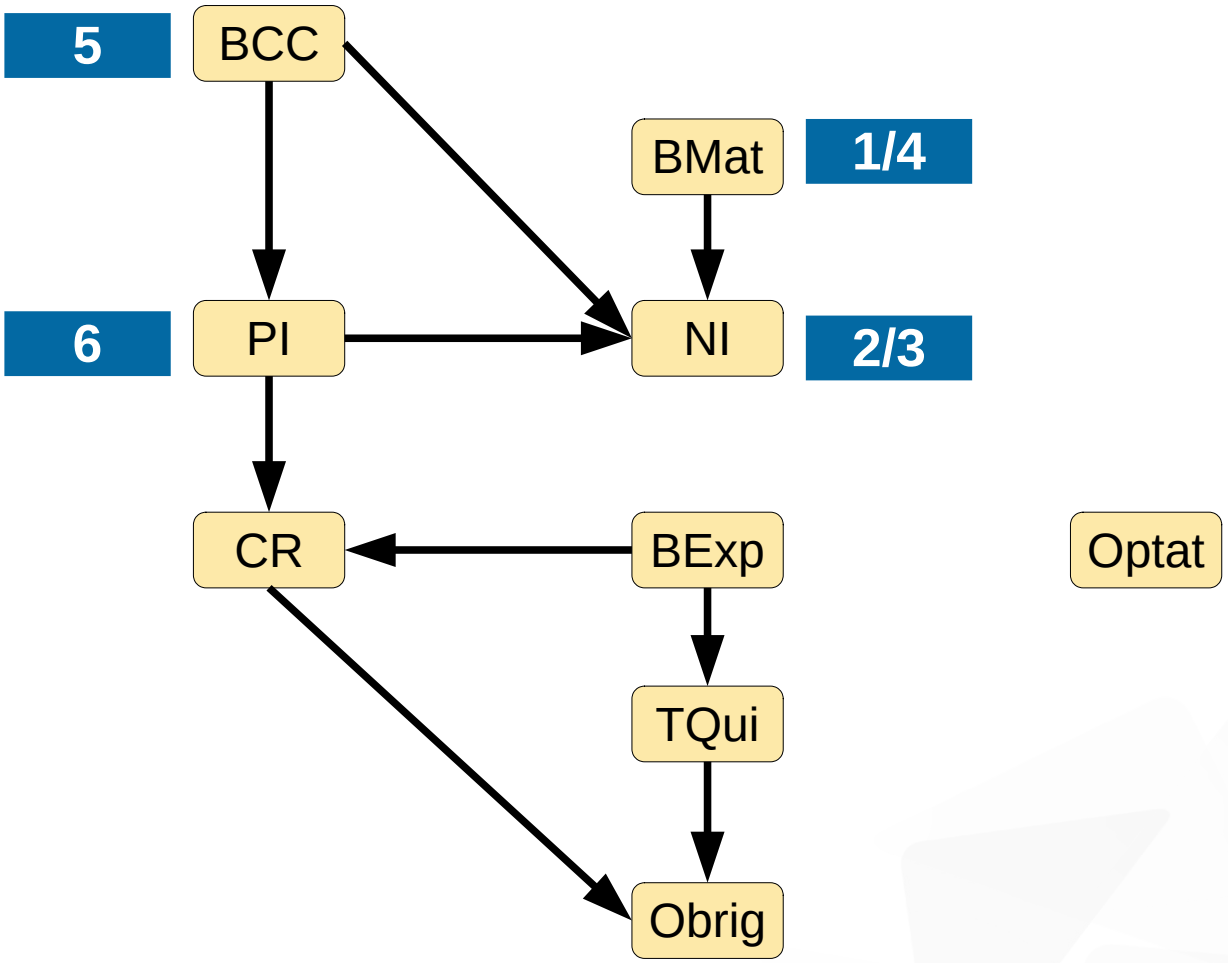
Ordenação 'topologica'



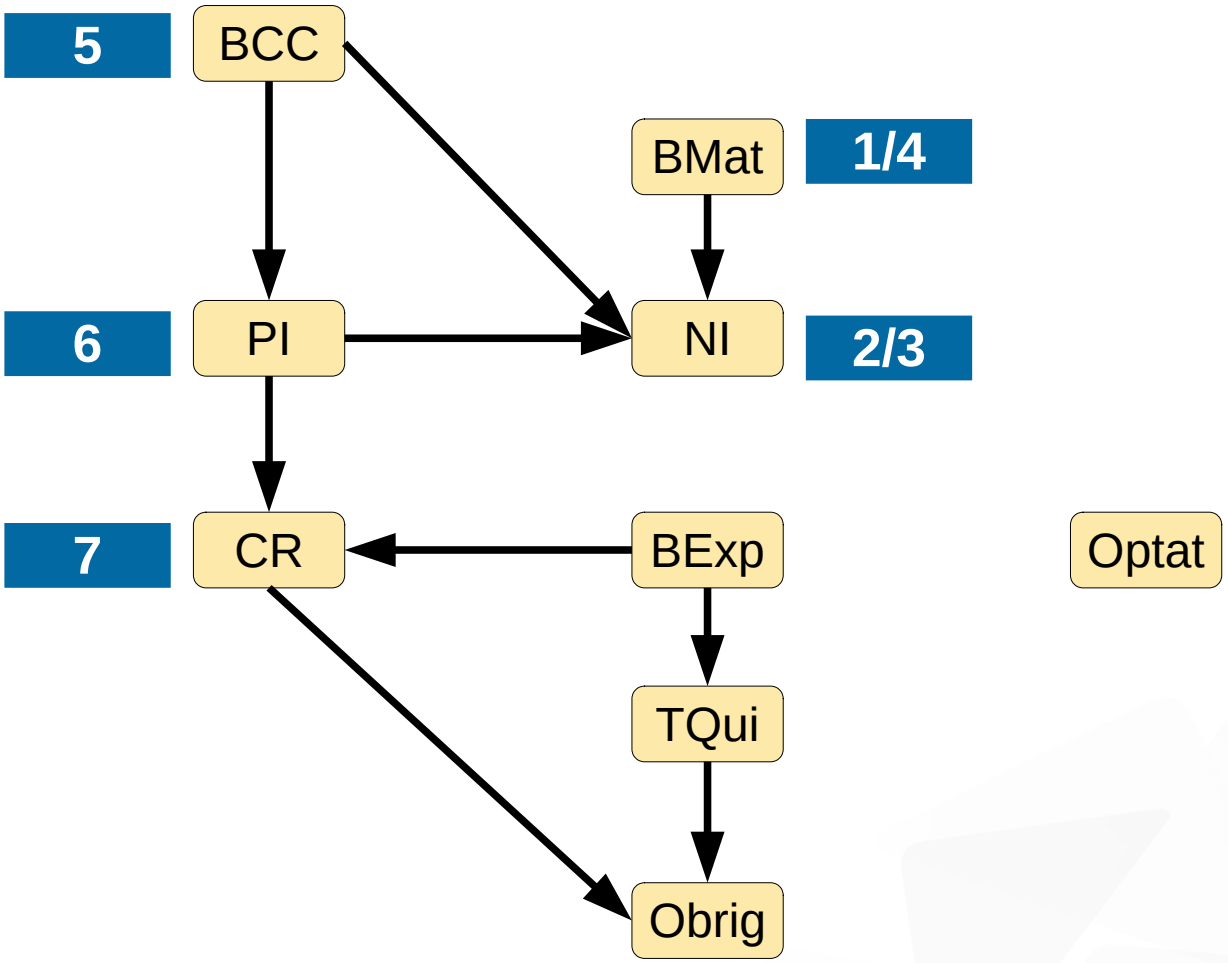
Ordenação 'topologica'



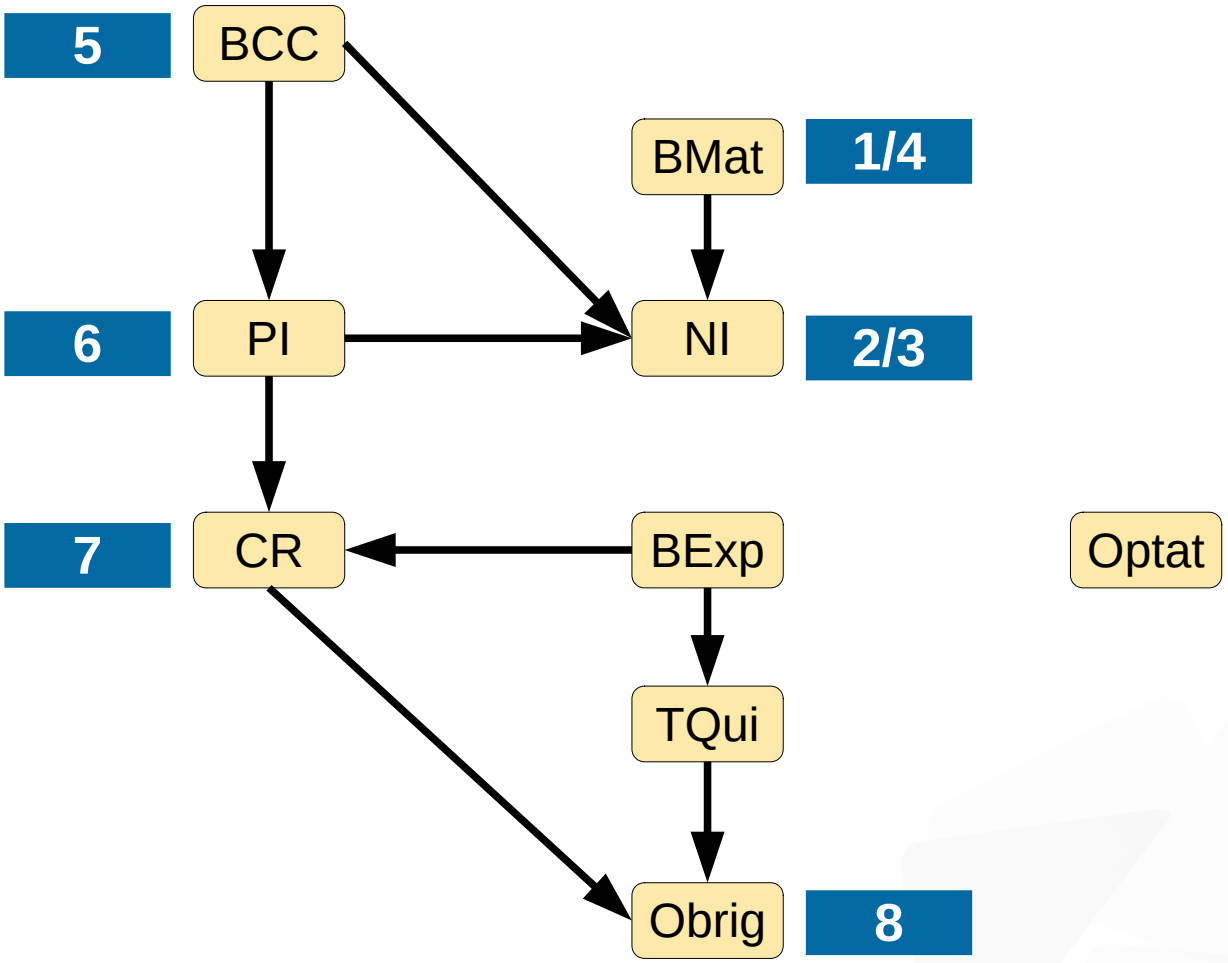
Ordenação 'topologica'



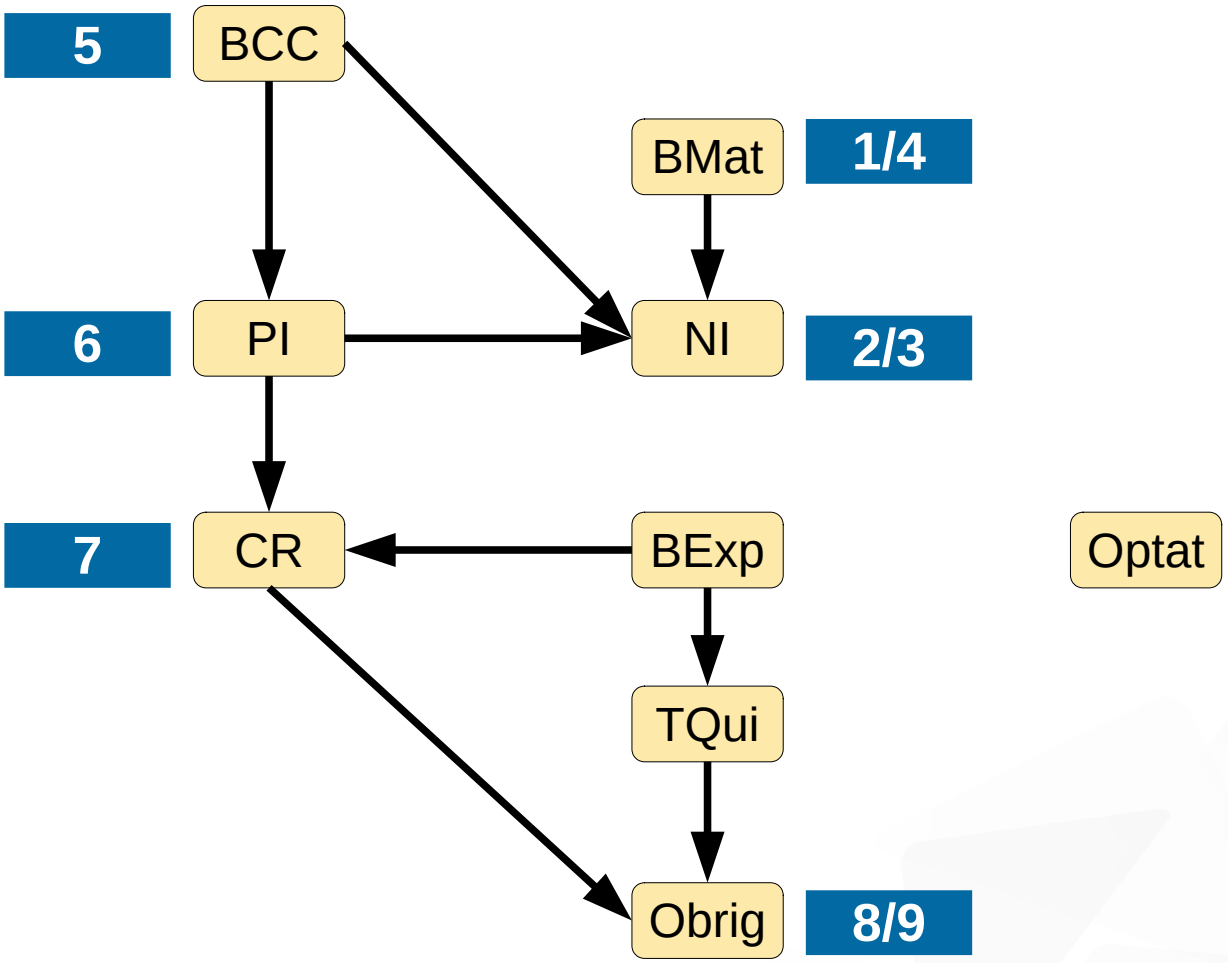
Ordenação 'topologica'



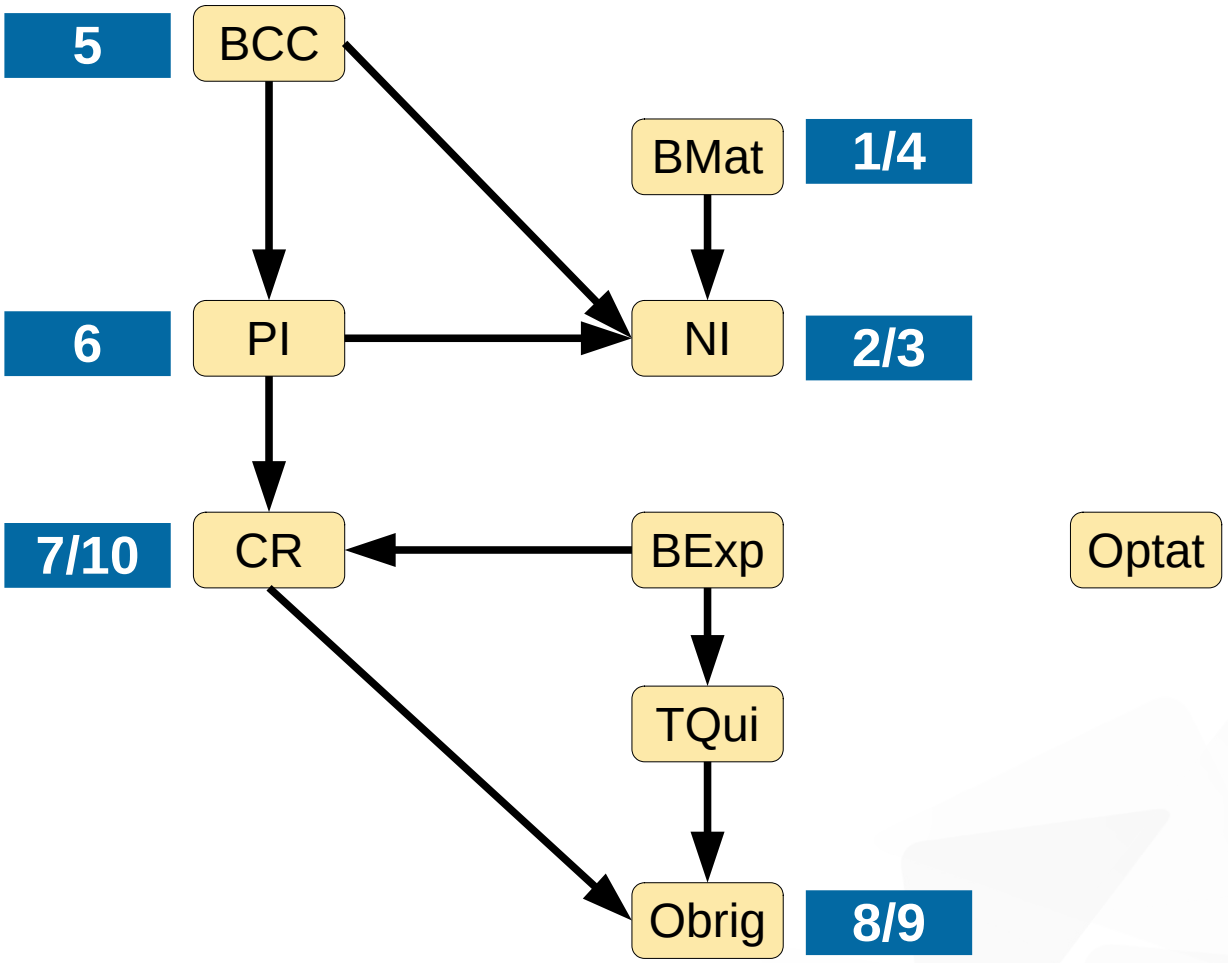
Ordenação 'topologica'



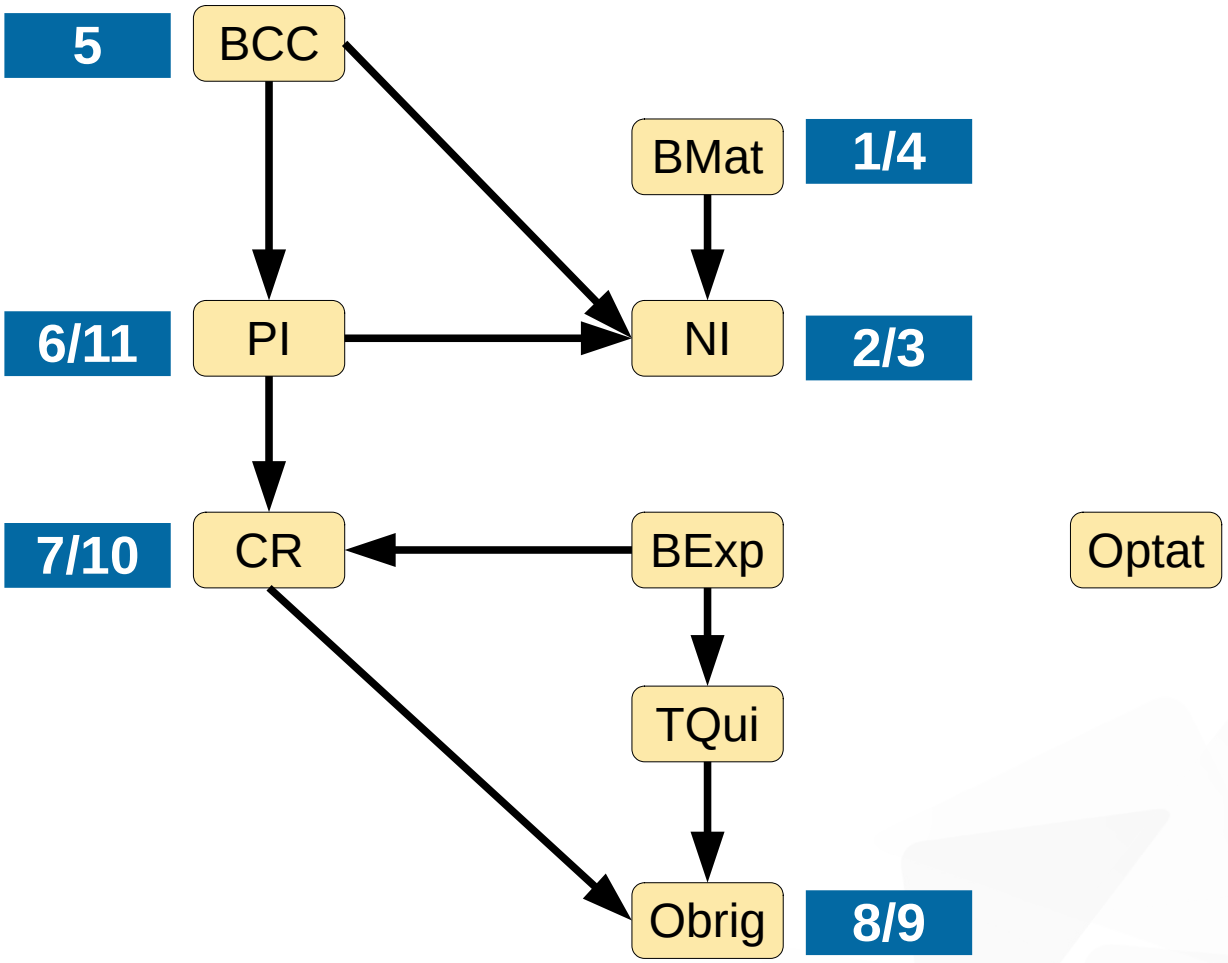
Ordenação 'topologica'



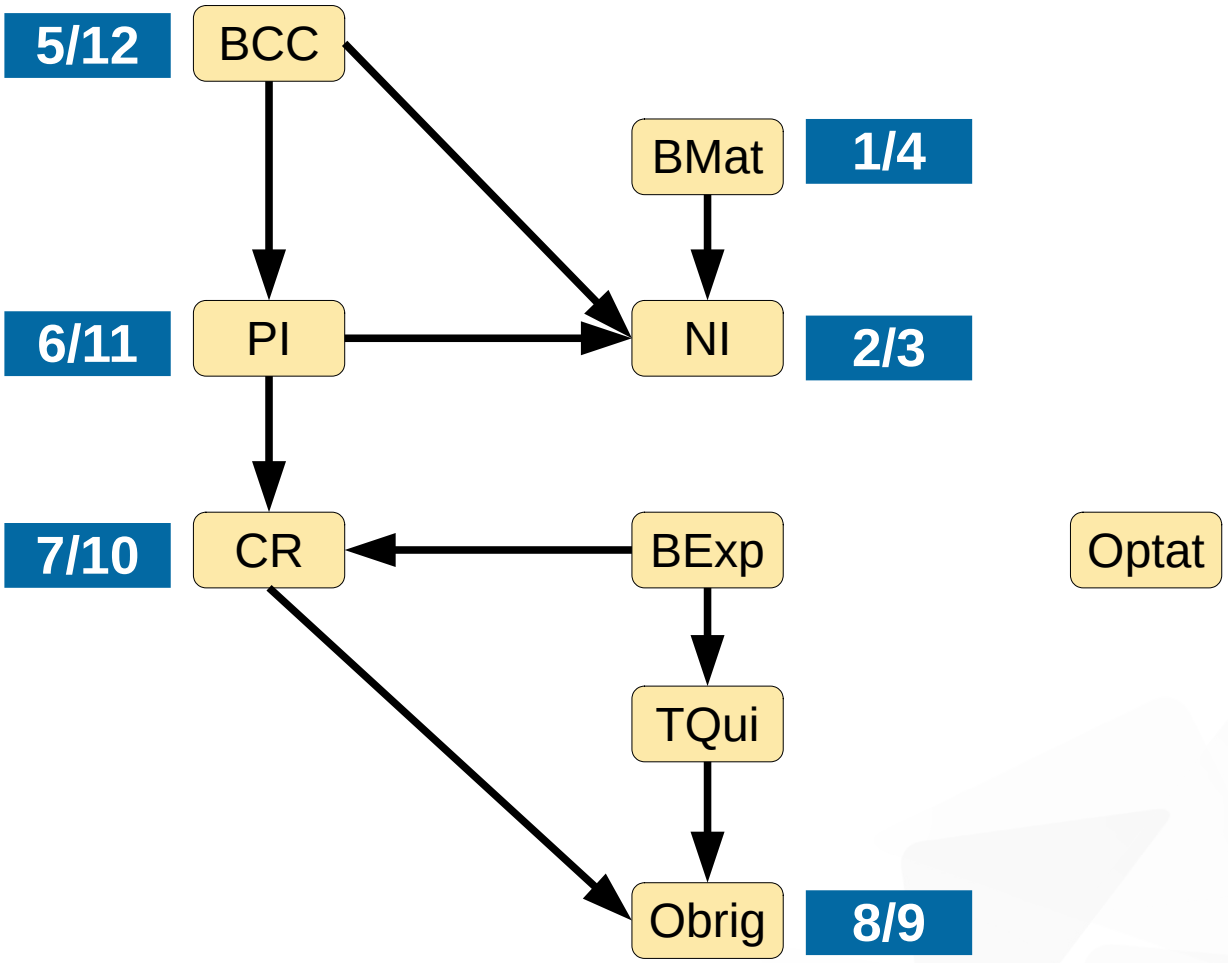
Ordenação 'topologica'



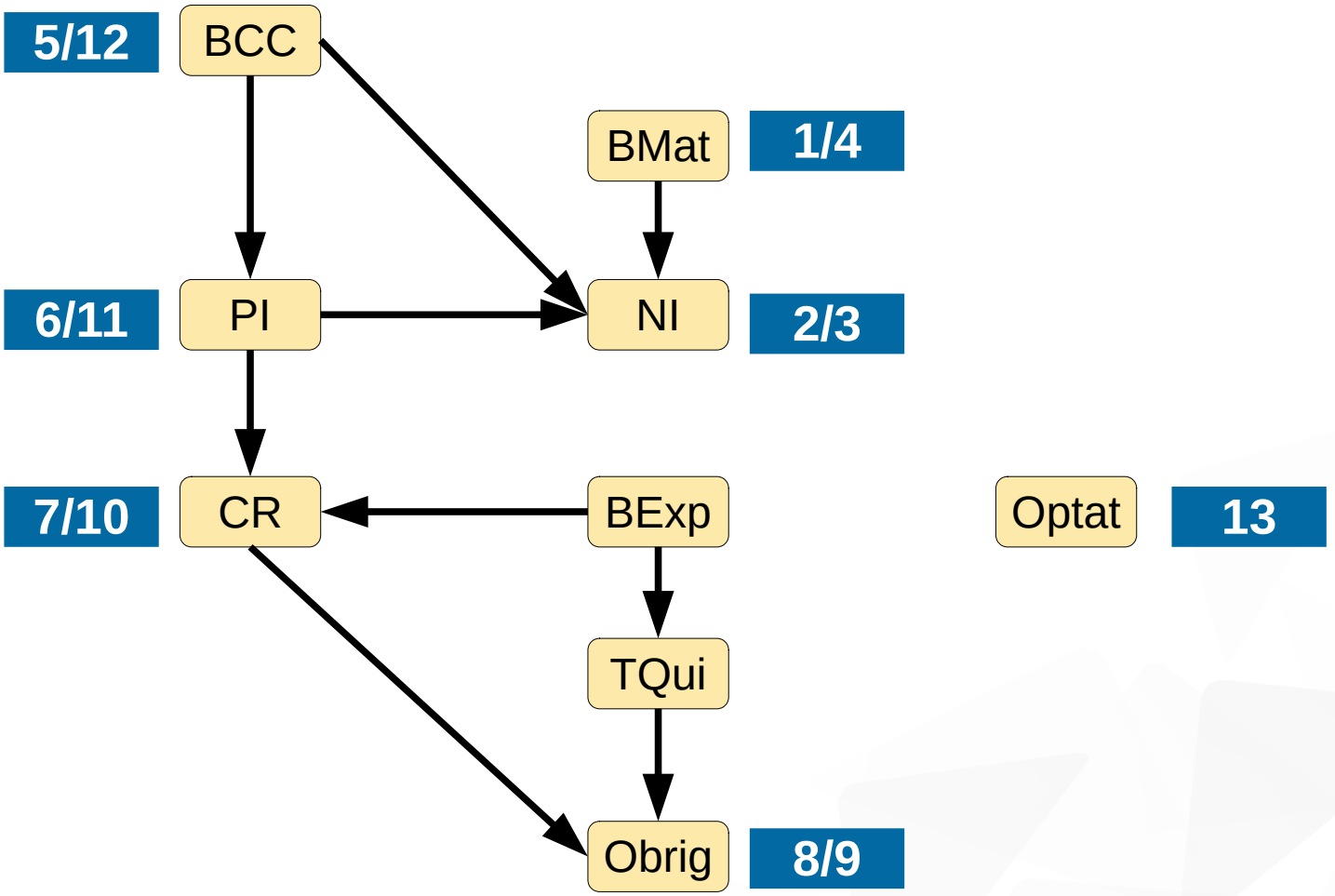
Ordenação 'topologica'



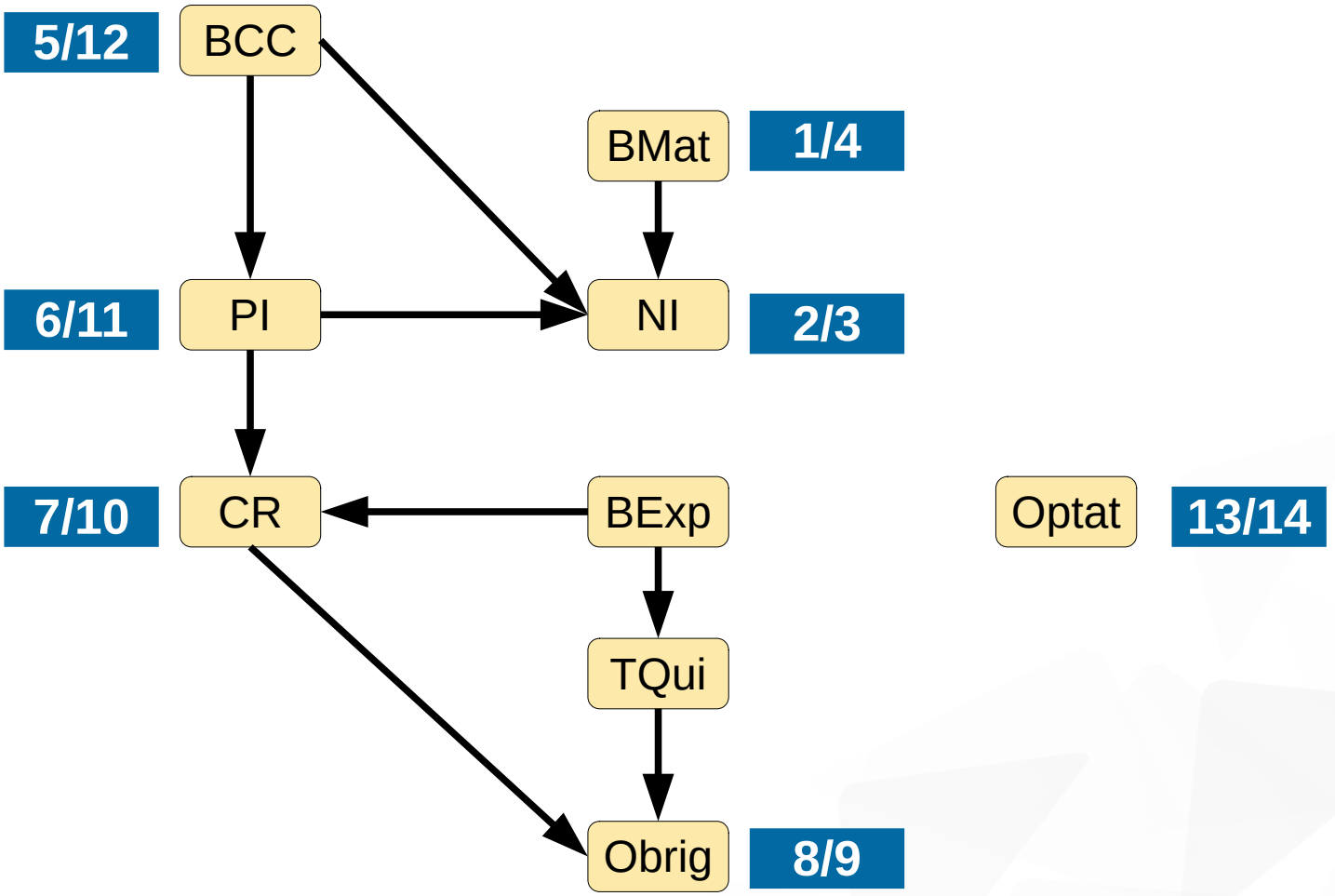
Ordenação 'topologica'



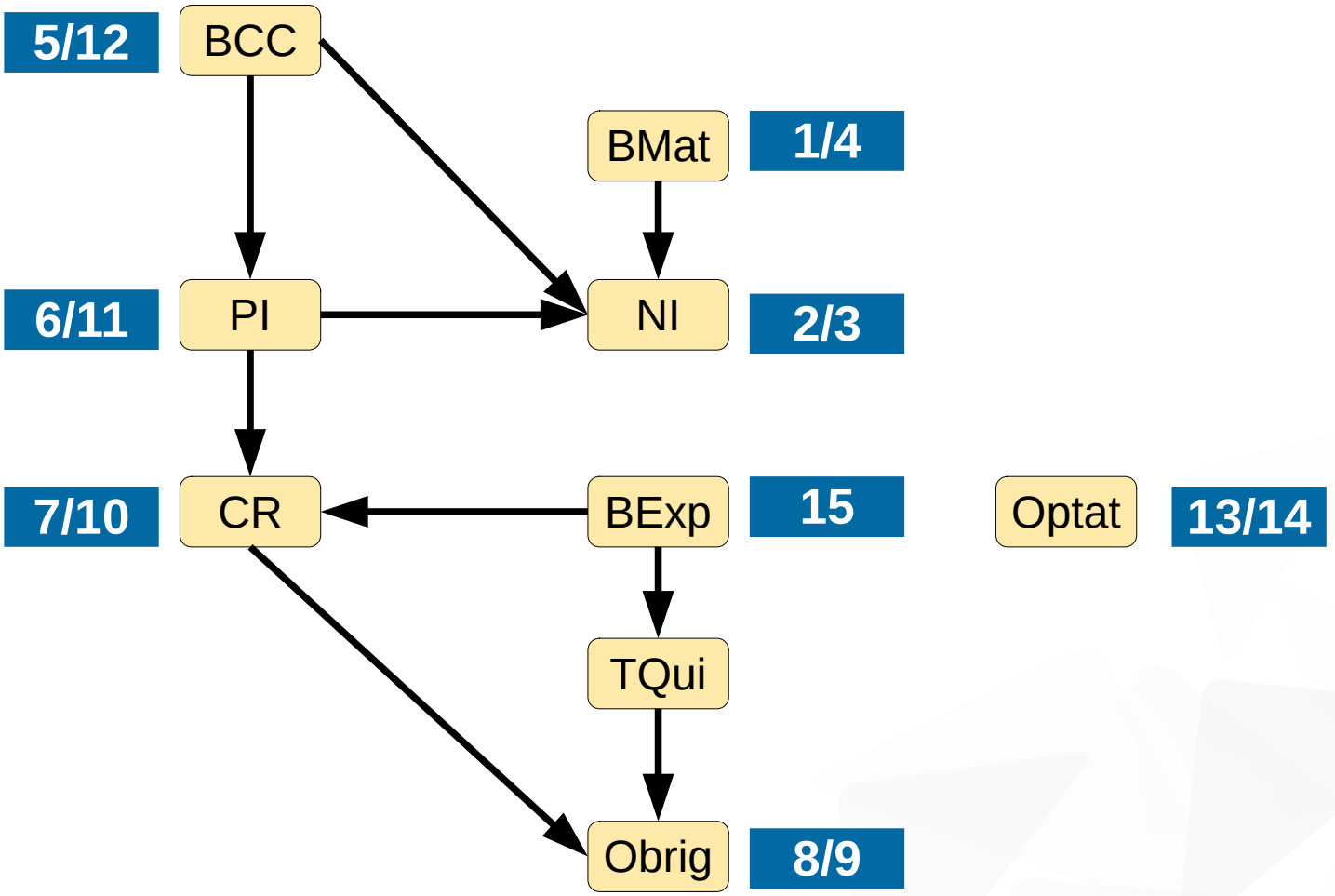
Ordenação 'topologica'



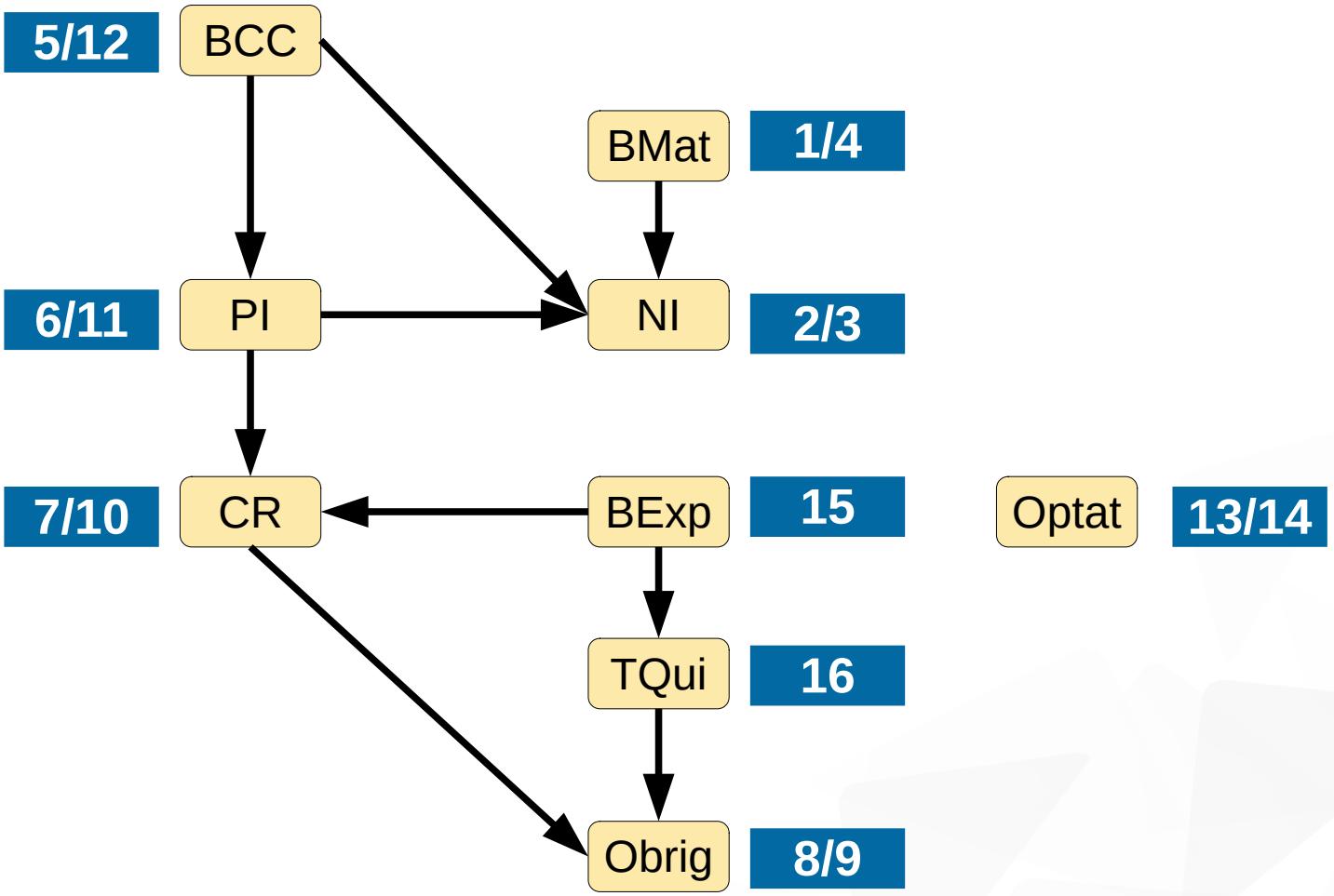
Ordenação 'topologica'



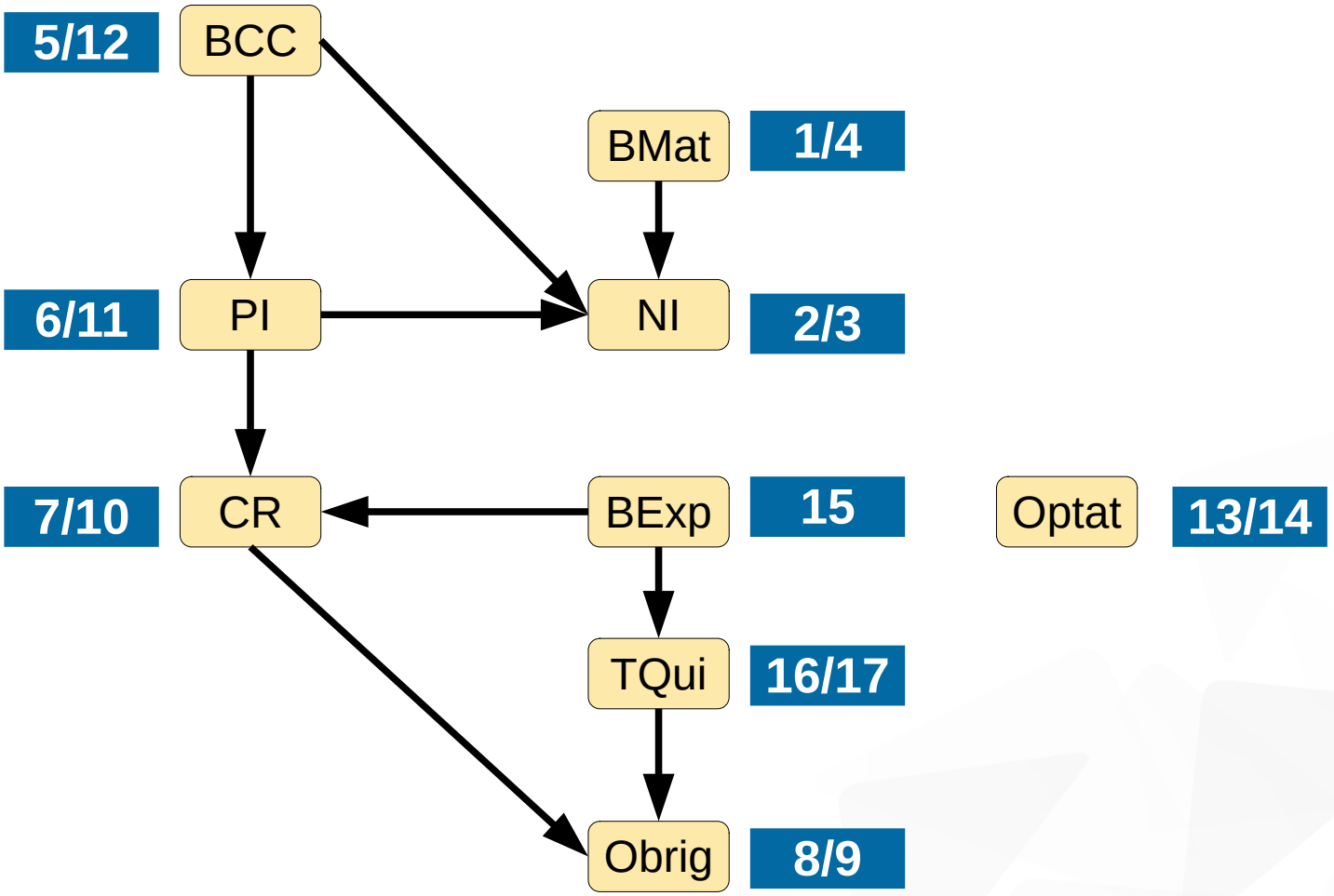
Ordenação 'topologica'



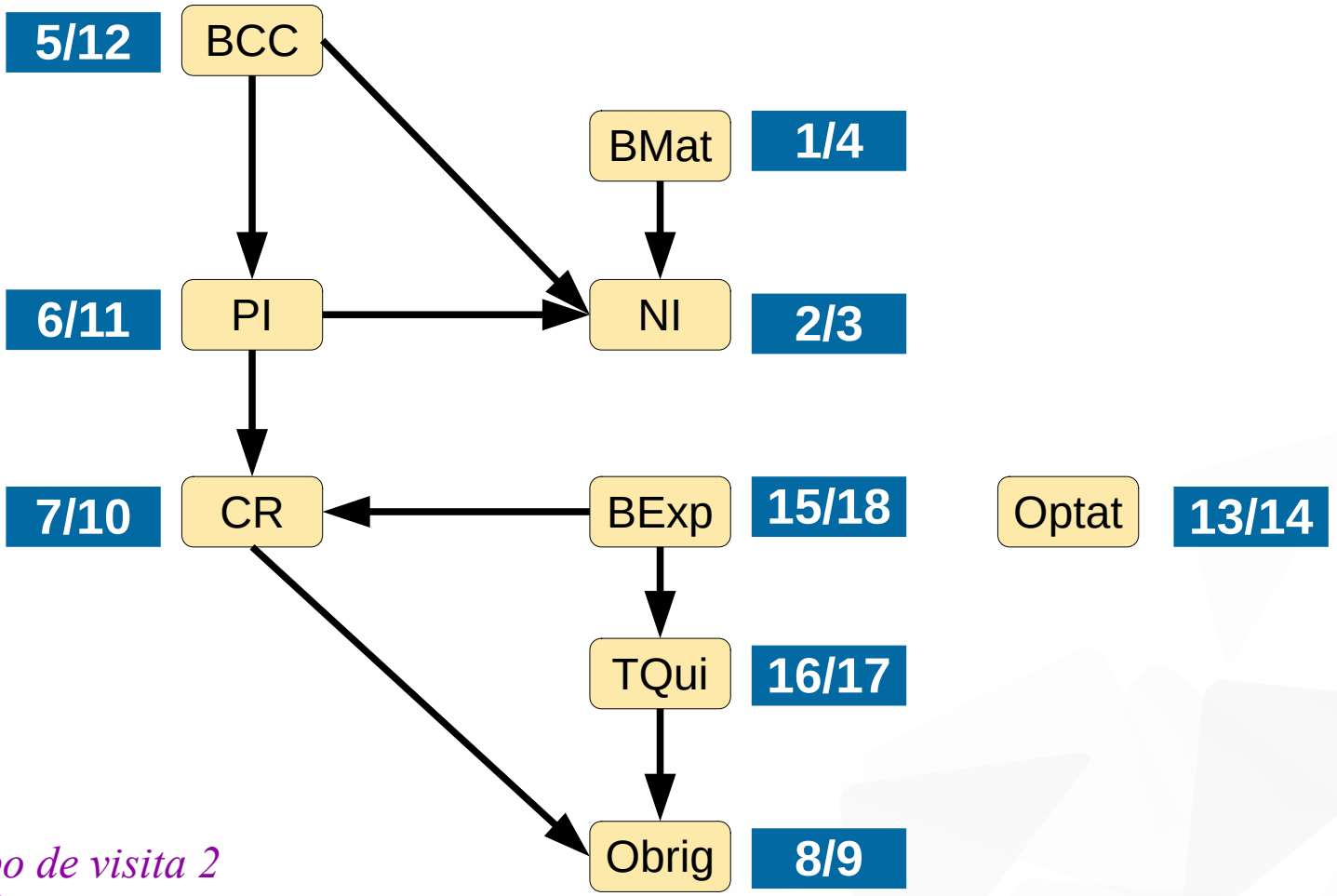
Ordenação 'topologica'



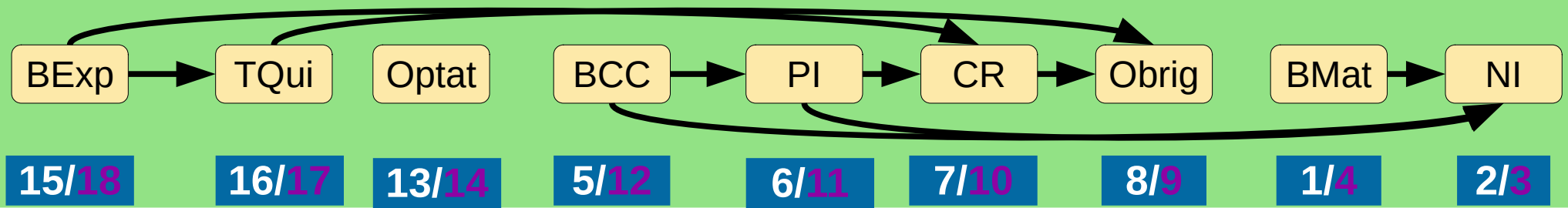
Ordenação 'topologica'



Ordenação 'topologica'



Ordenado pelo tempo de visita 2 (ordem descendente)



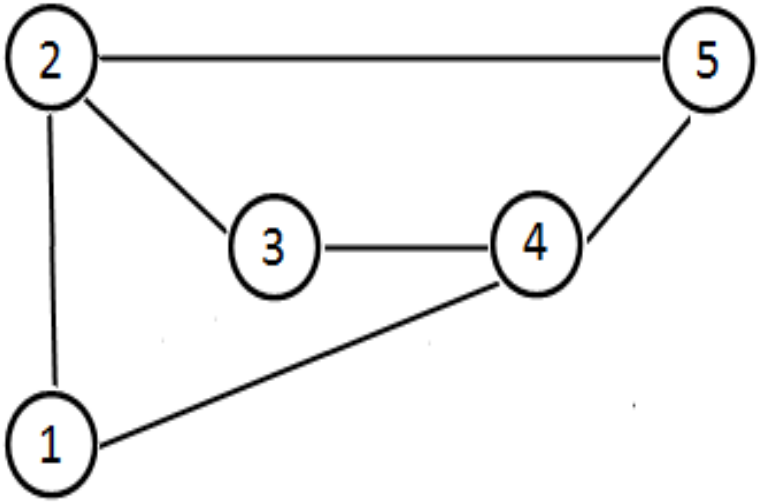
III. Atividade Prática

Atividade Prática

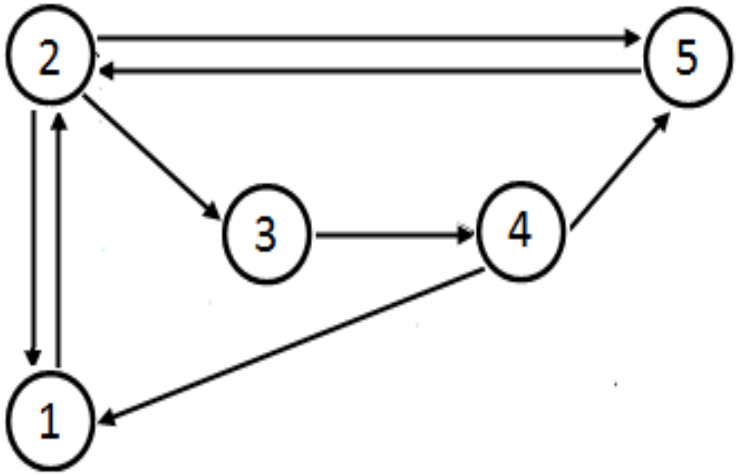
1. Para os grafos G e H abaixo, execute a **busca em profundidade** a partir do **vértice 1**:

- (a) Dando preferência para vértices de **menor** índice.
- (b) Dando preferência para vértices de **maior** índice.

Para cada exercício indique a sequência de vértices visitados.



G



H

Atividade Prática

Grafo G:

- (a) Dando preferência para vértices de **menor** índice.
  <1,2,3,4,5>

- (b) Dando preferência para vértices de **maior** índice.
  <1,4,5,2,3>

Grafo H:

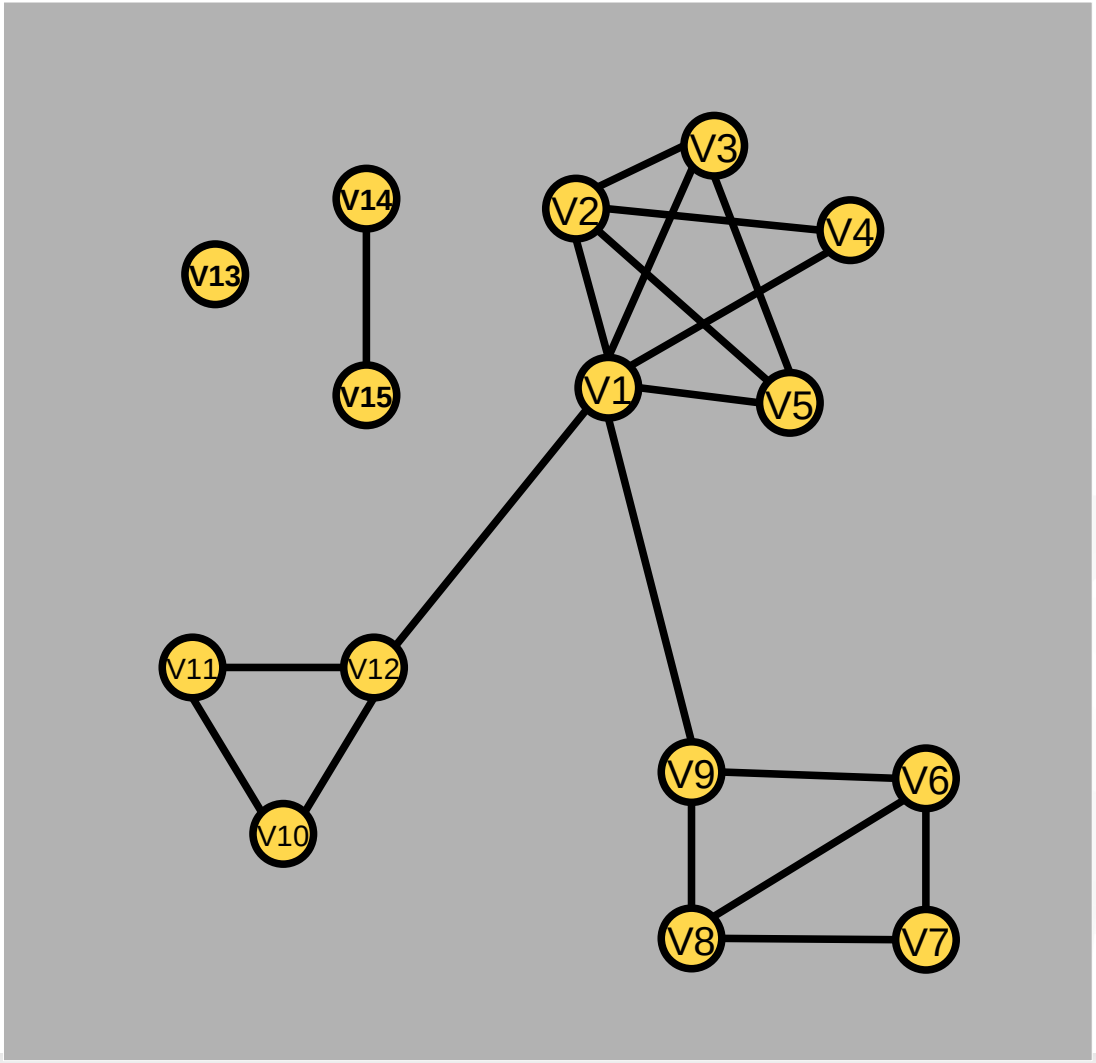
- (a) Dando preferência para vértices de **menor** índice.
  <1,2,3,4,5>

- (b) Dando preferência para vértices de **maior** índice.
  <1,2,5,3,4>

Atividade Prática

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

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