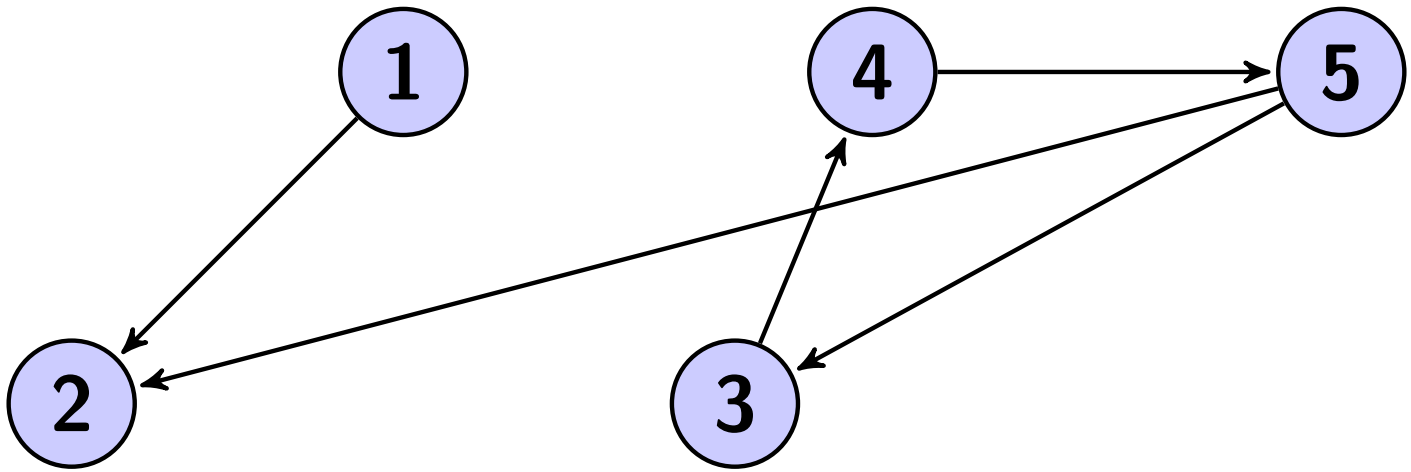
Dígrafos, arestas com pesos e menor caminho

Marcelo K. Albertini

25 de fevereiro de 2014

Grafo direcionado ou **Dígrafo**

Conjunto de vértices conectados por arestas direcionadas.



Em dígrafo existe **grau de saída** e de **grau de entrada**: grau de saída de 5 é 2 e grau de entrada de 5 é 1.

Aplicações de dígrafos

dígrafo	**vértice**	**aresta direcionada**
transportes	cruzamentos	ruas de mão-única
web	páginas web	links
cadeia alimentar	espécies	predador-presa
escalonamento	tarefa	restrição de precedência
finanças	banco	transação
celulares	pessoa	ligação
DST	pessoa	infecção
jogos	posição tabuleiro	movimento
literatura científica	artigo científico	citação
coletor de lixo	**objeto**	**ponteiro**

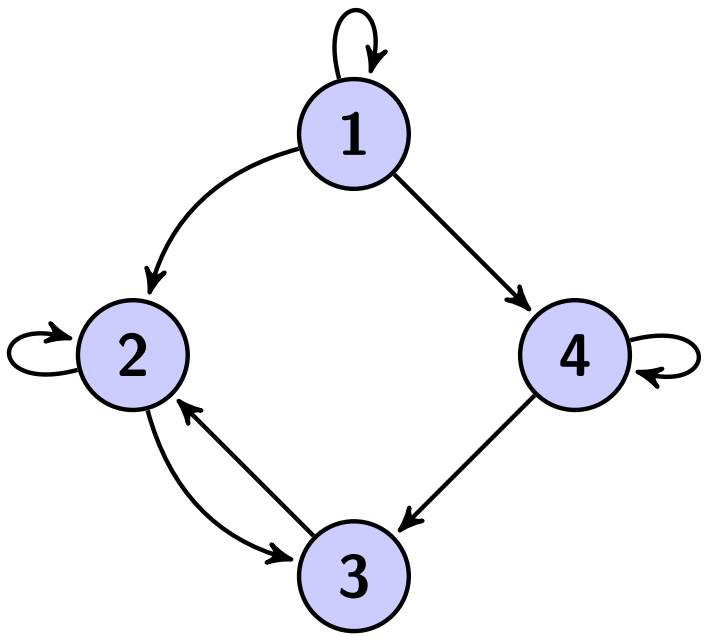
API de Dígrafos

API similar à de Grafos

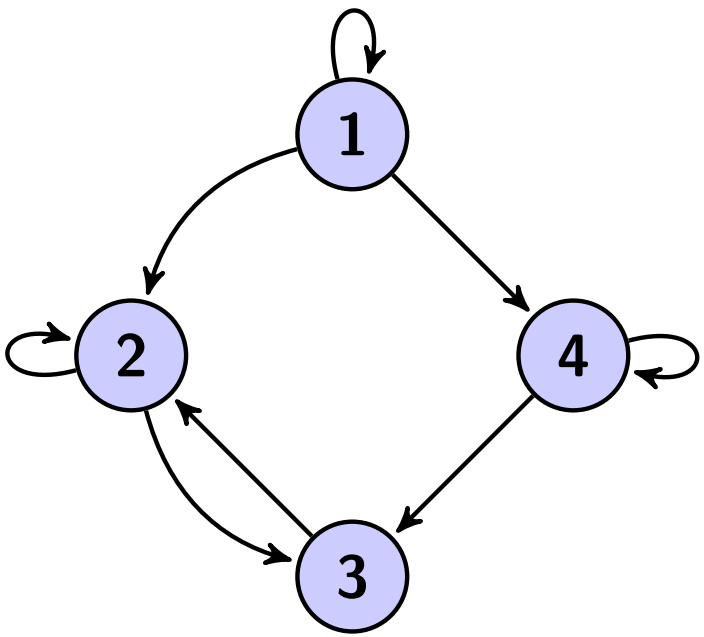
public class	**Digrafo**	
	Digrafo(int V)	criar dígrafo vazio com V vértices
	Digrafo(InputStream in)	criar dígrafo a partir de entradas
void	novaAresta(int v, int w)	criar aresta
Iterable<Integer>	adj(int v)	vértices adjacentes a v
int	V()	número de vértices
int	E()	número de arestas
Digrafo	reverter()	digrafo com arestas reversas

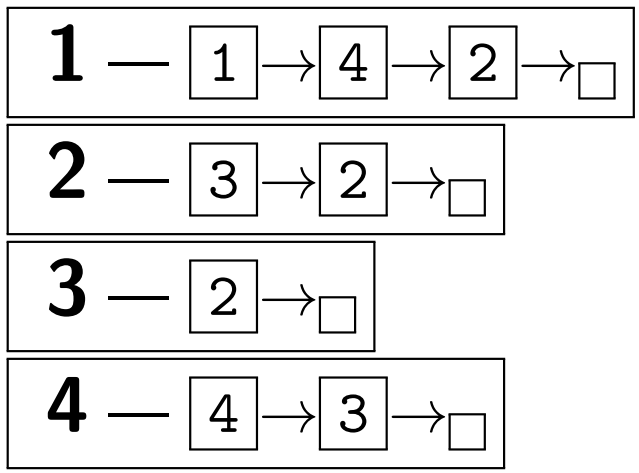
Problemas de dígrafos

- Caminho. Existe um caminho direcionado de `s` para `t`?
- Menor caminho. Qual é o menor caminho direcionado de `s` a `t`?
- Ordenação topológica. É possível organizar o dígrafo de tal forma que nenhuma aresta aponta para baixo?
- Conectividade forte. Existe um caminho direcionado entre todos os pares de vértices?
- Cerco transitivo. Para quais vértices `v` e `w` existe um caminho de `v` para `w`?
- PageRank. Qual é a importância de uma página web?



Representação de um dígrafo com listas de adjacências





Implementação muito similar ao `Grafo`, porém ao inserir aresta, só define o sentido pedido e não os dois.

Problema: alcançabilidade

Problema

Encontrar todos os vértices alcançáveis a partir de s.

Problema: alcançabilidade

Problema

Encontrar todos os vértices alcançáveis a partir de s.

Solução

Busca em profundidade em dígrafos. O código é igual ao utilizado em grafos.

Aplicação de alcançabilidade

Aplicação

Análise do fluxo de controle de programas para verificação de `bugs`.

- Código é representado por dígrafo
  - Vértice: bloco de instruções sem desvios
  - Aresta: desvio do fluxo de operações (`if`, `for`, `while`, `função`, `...`)
- É possível detectar
  - Regiões mortas - nunca alcançáveis
  - Loops infinitos - nunca é possível sair

Coletor de lixo

Cada estrutura de dados é um dígrafo

- Vértice = objeto (struct)
- Aresta = referência (ponteiro)

- Pontos de partida
  - objetos diretamente acessáveis ao programa (exemplo: pilha)
- Objetos alcançáveis
  - indiretamente alcançáveis por referências
- Objetos não alcançáveis podem ser liberados

Algoritmo marcar-limpar

- Marcar-limpar (McCarthy, 1960)
  - Marcar todos os objetos alcançáveis
  - Limpar objetos não marcados (lixo)
- Custo do algoritmo
  - 1 bit extra de marcação por objeto
  - memória temporariamente usada pela busca em profundidade

Problema do menor caminho

- Routeamento em mapas
- Navegação de robôs
- Planejamento de tráfico urbano
- Planejamento de atividades em uma empresa
- Algoritmos de protocolos de roteamento
- Redimensionamento inteligente de imagens

É necessário

- Representar pesos em arestas no dígrafo
- Caminhar no dígrafo e atualizar distância
  - ordem topológica
  - algoritmo de Dijkstra

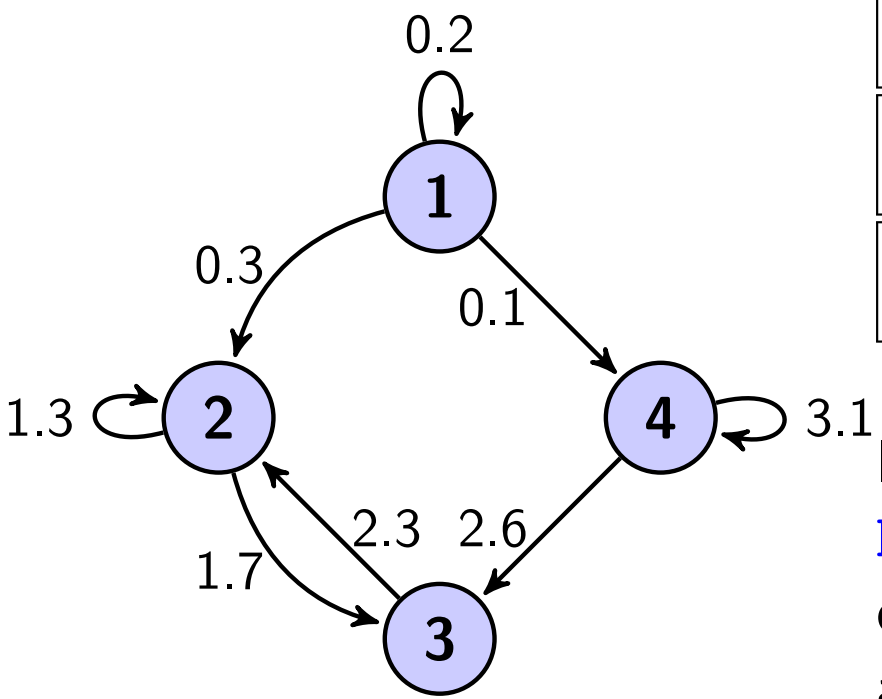
API: Arestas com pesos

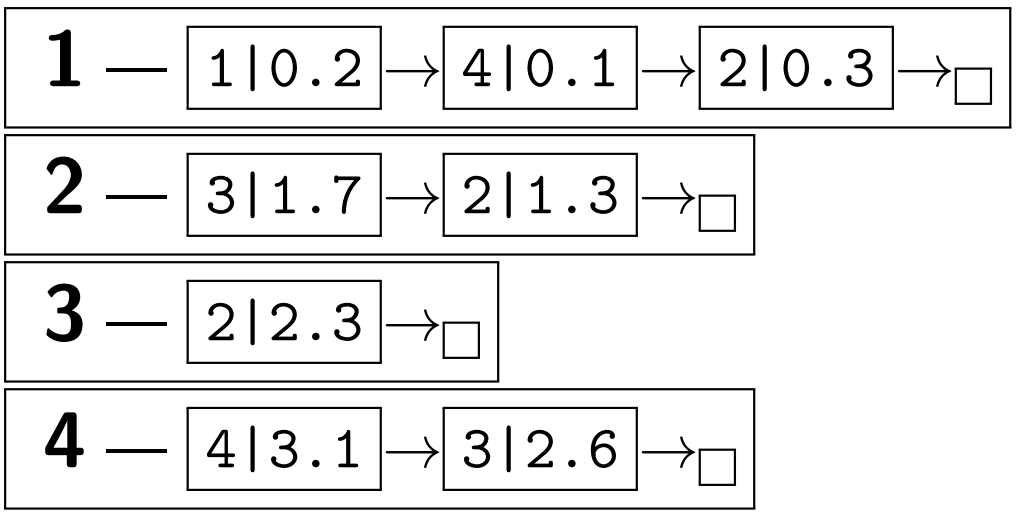
class	**Aresta**	implements Comparable<Aresta>
	Aresta(int v, int w, double peso)	criar aresta v-w com peso
int	de()	vértice origem da aresta
int	para()	vértice destino da aresta
double	peso()	peso associado
int	compareTo(Aresta a)	compara pesos

API de Dígrafos com pesos

public class	**DigrafoComPeso**	
	DigrafoComPeso(int V)	criar dígrafo vazio com V vértices
void	novaAresta(Aresta a)	criar aresta
Iterable<Aresta>	adj(int v)	vértices adjacentes a *v*

Representação de um dígrafo com listas de adjacências com pesos

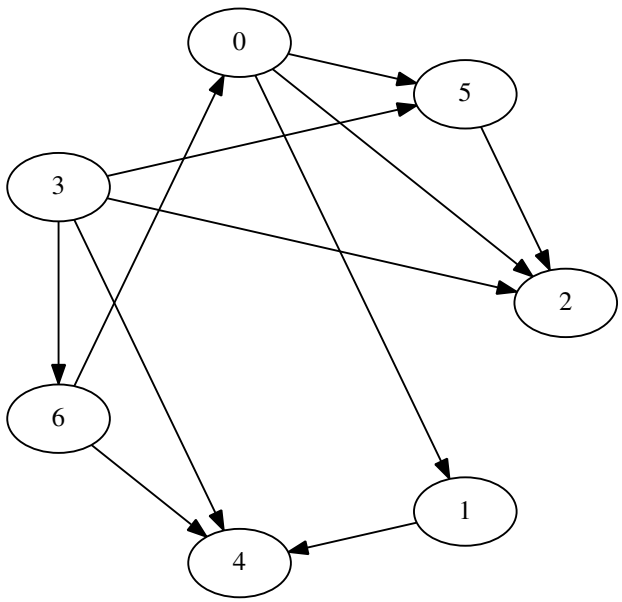


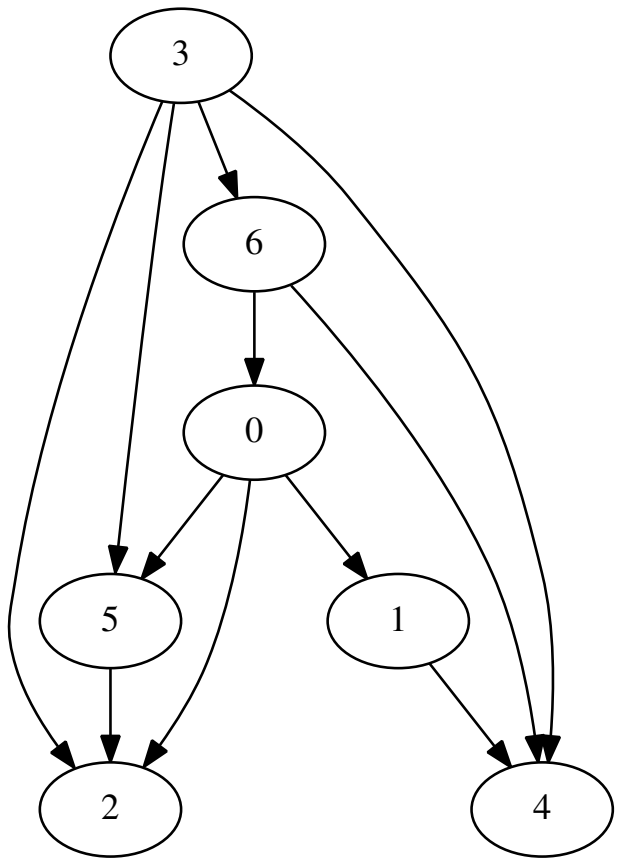


Implementação muito similar ao `Dígrafo`, porém ao inserir aresta, deve-se `criar uma aresta com peso` antes.

Ordenação topológica

- Problema da precedência
  - Dado um cronograma, quais tarefas posso fazer antes?
  - Dados disciplinas e pré-requisitos, qual ordem de disciplinas posso fazer?





Ordenação topológica

Problema

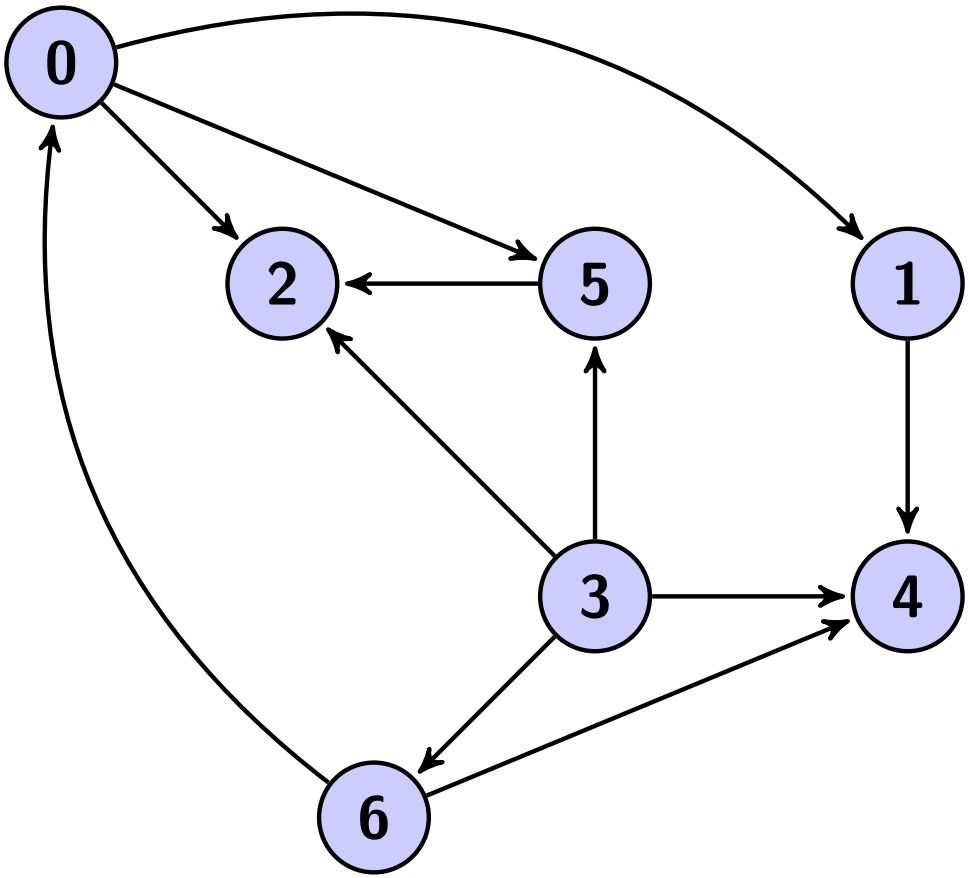
- Temos um grafo direcionado sem ciclos
- Objetivo: Redesenhar grafo, tal que todas as arestas apontam para o mesmo sentido.

Solução

- Usar busca em profundidade.
- Retornar vértices em pós-ordem reversa
  - Pós-ordem ocorre após usar cada vértice

```
1  class OrdemTopologica {
2    boolean[] marcado;
3    Stack<Integer> posOrdemRev;// ordem topologica
4
5    public OrdemTopologica(Digrafo G) {
6     posOrdemRev = new Stack<Integer>();
7     marcado = new boolean[G.V()];
8     for (int v = 0; v < G.V(); v++)
9       if (!marcado[v]) dfs(G, V);
10   }
11
12   void dfs(Digrafo G, nt v) { // busca em profundidade
13     marcado[v] = true;
14     for (int w : G.adj(v))
15       if (!marcado[w]) dfs(G, w);
16     posOrdemRev.push(v); // poe no topo da pilha
17   }
18
19   public Iterable<Integer> ordem() {
20     return posOrdemRev;
21   }
22 }
```

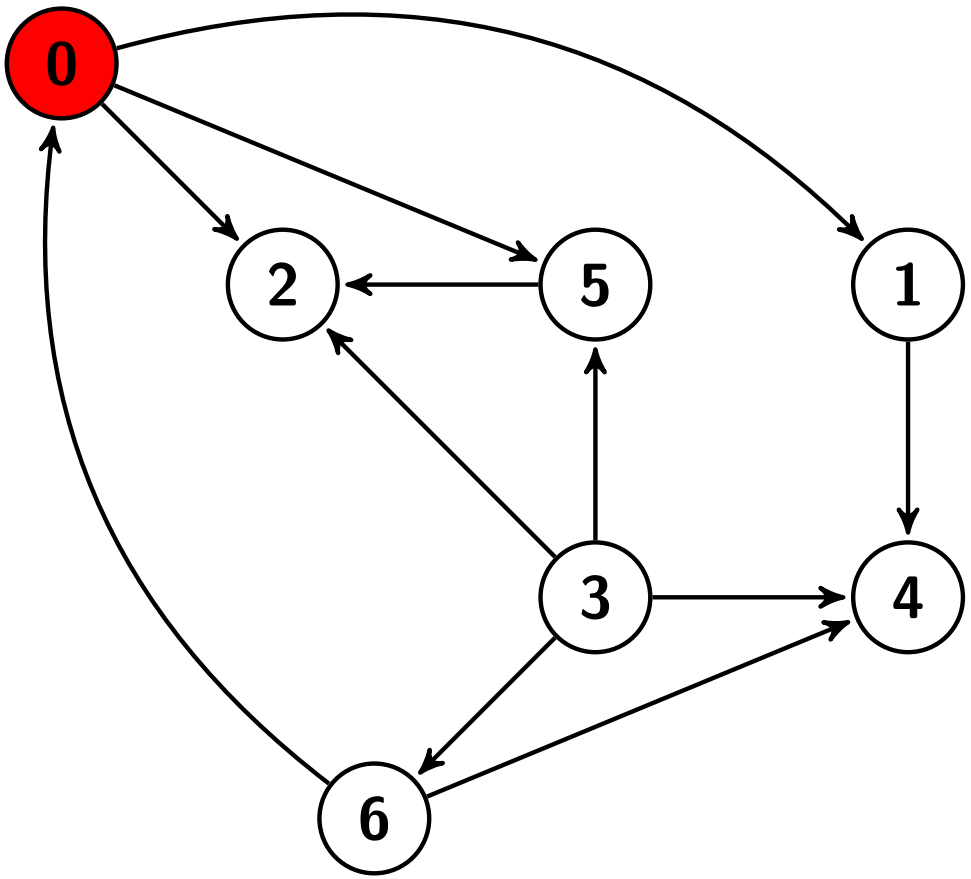
Ordenação topológica



$0 \to 5$
$0 \to 2$
$0 \to 1$
$3 \to 6$
$3 \to 5$
$3 \to 4$
$5 \to 4$
$6 \to 4$
$6 \to 0$
$3 \to 2$
$1 \to 4$

Um grafo direcionado sem ciclos.

Ordenação topológica



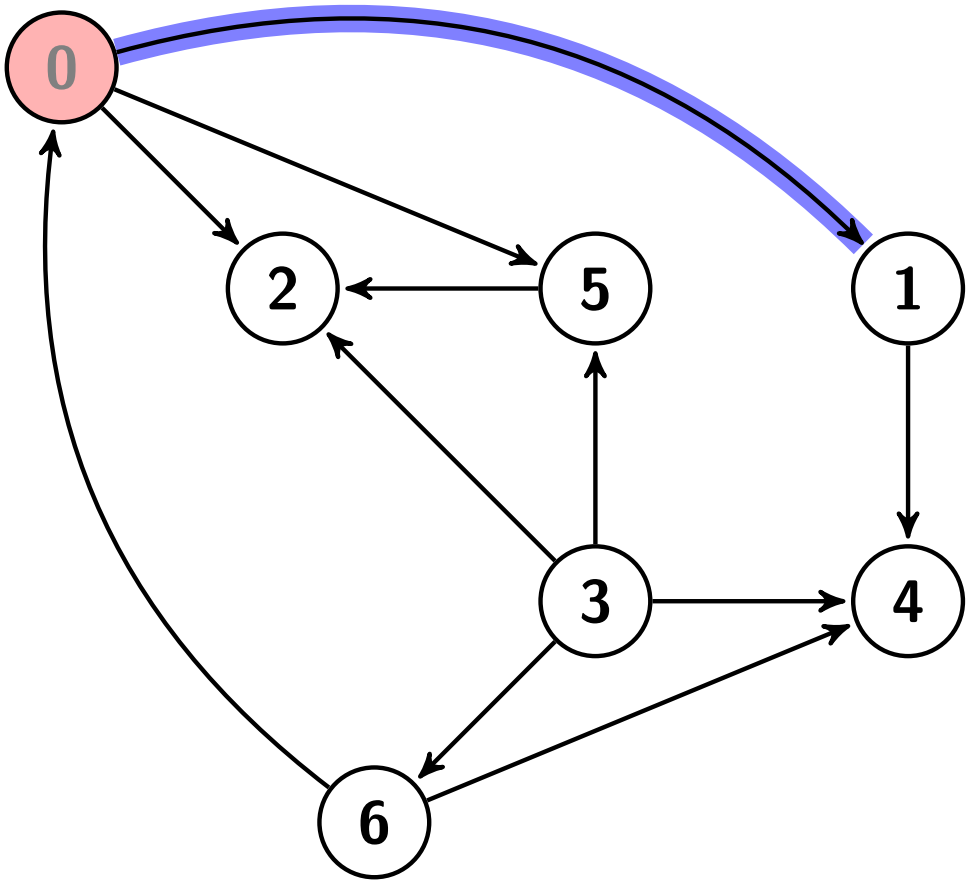
Pós-ordem - pilha

- {}

Ação

visita 0; avaliar 1; avaliar 2; e avaliar 5

Ordenação topológica



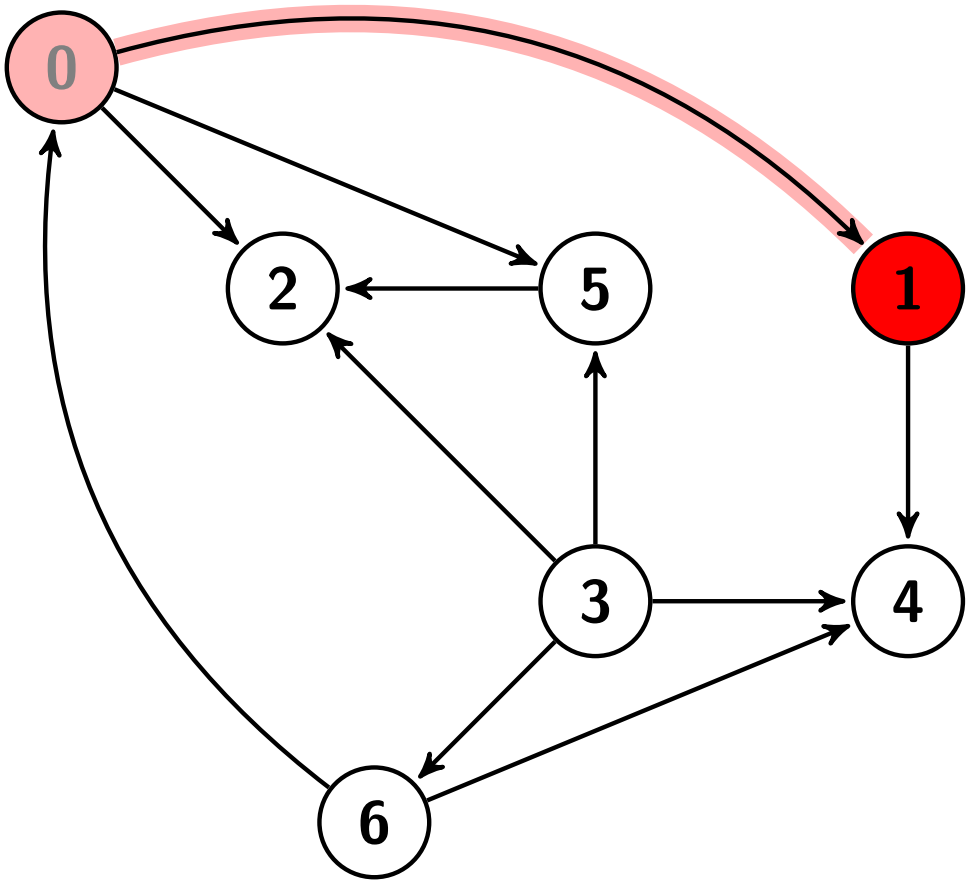
Pós-ordem - pilha

- {}

Ação

visita 0; avaliar 1; avaliar 2; e avaliar 5

Ordenação topológica



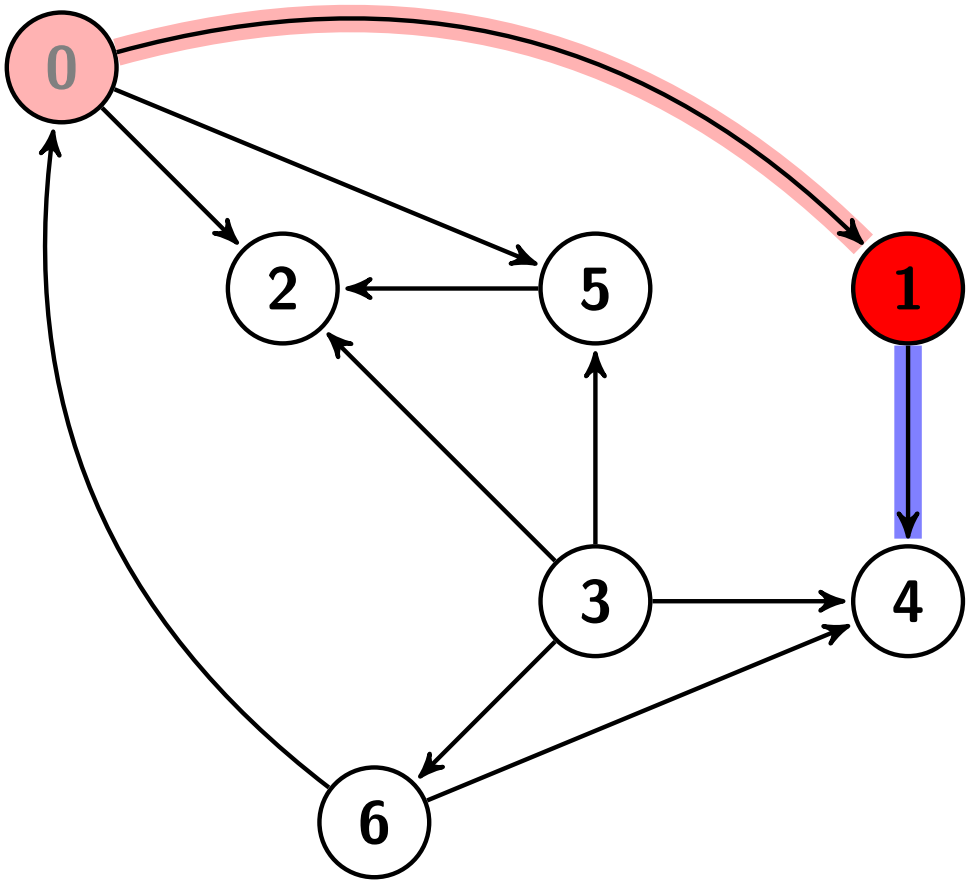
Pós-ordem - pilha

- {}

Ação

visita 1; avaliar 4;

Ordenação topológica



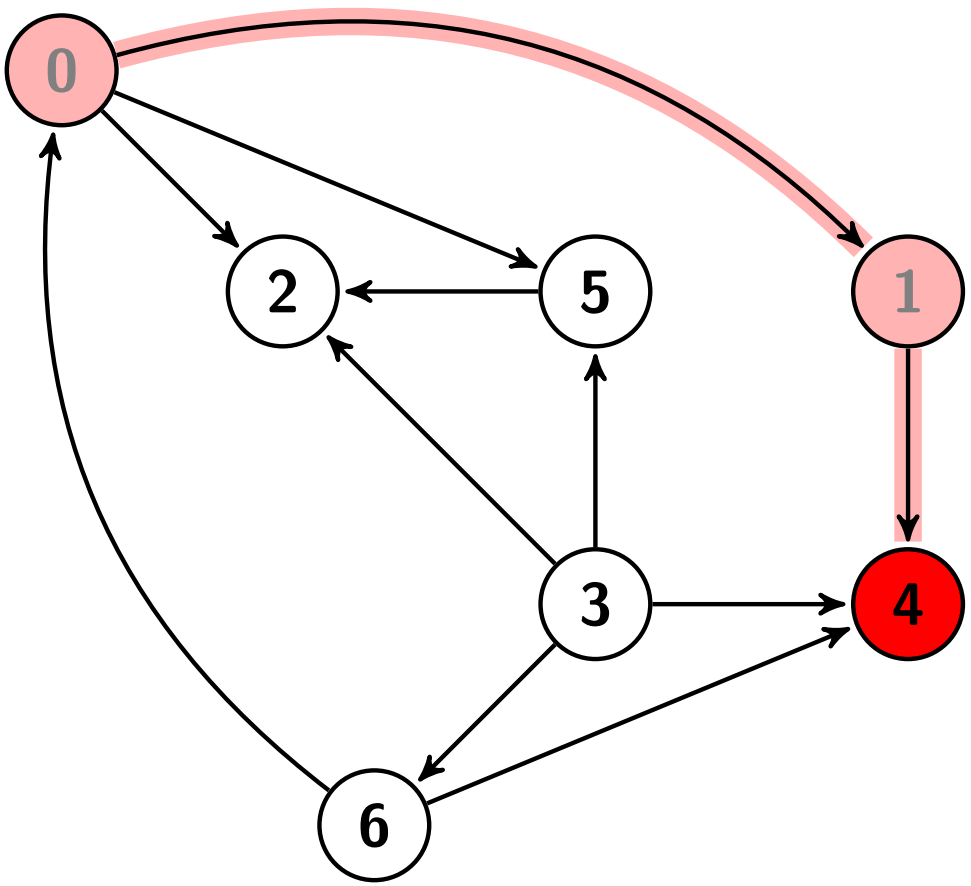
Pós-ordem - pilha

- {}

Ação

visita 1; avaliar 4;

Ordenação topológica



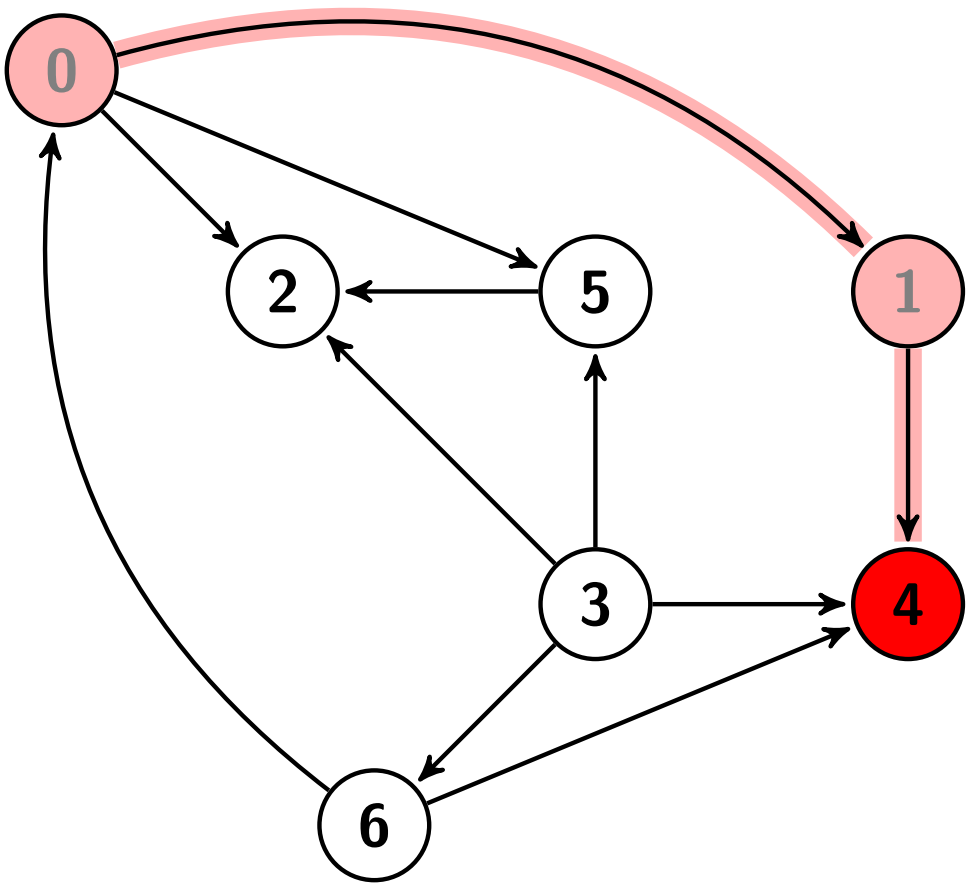
Pós-ordem - pilha

- {}

Ação

- visita 4;

Ordenação topológica



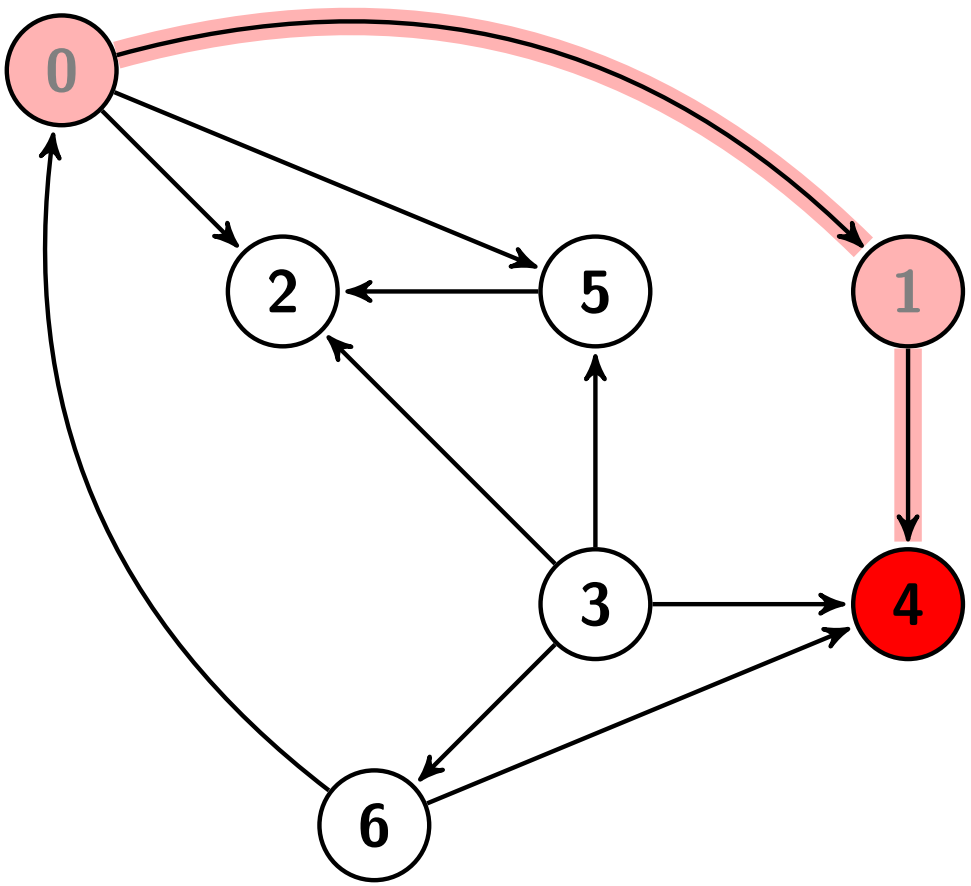
Pós-ordem - pilha

- {}

Ação

- visita 4;
- Não há adjacentes para avaliar

Ordenação topológica



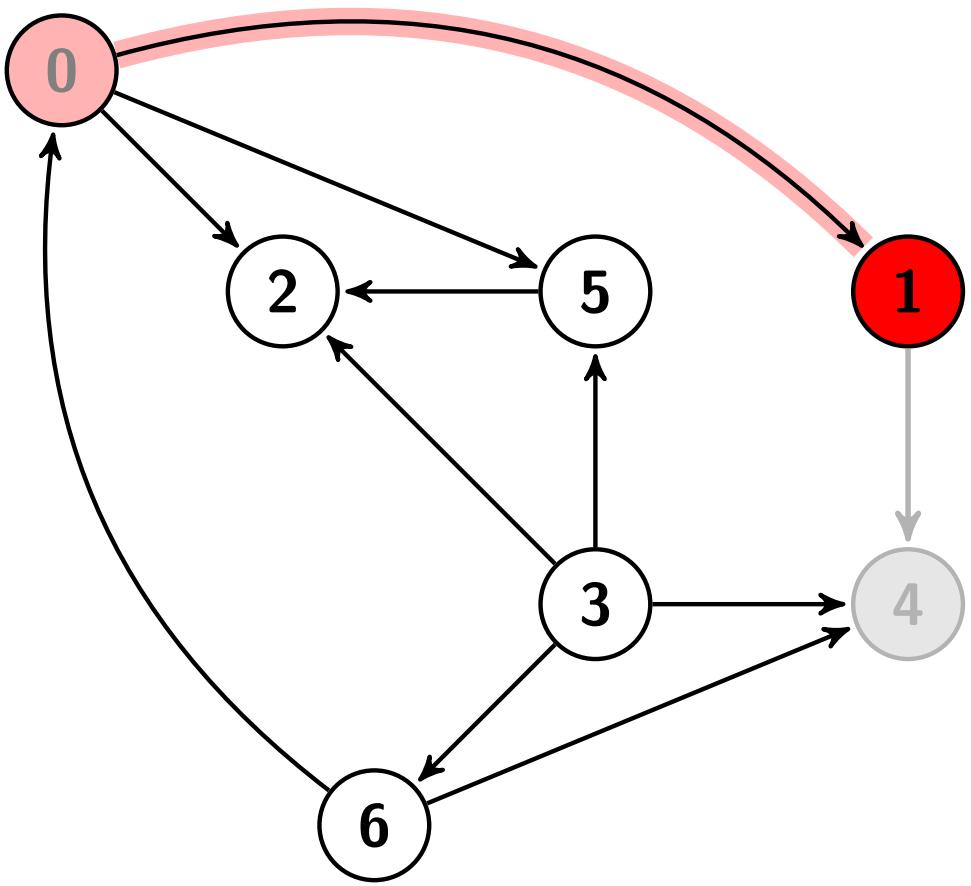
Pós-ordem - pilha

- {4}

Ação

- Não há adjacentes para avaliar
- Adicionar no caminho em pós-ordem

Ordenação topológica



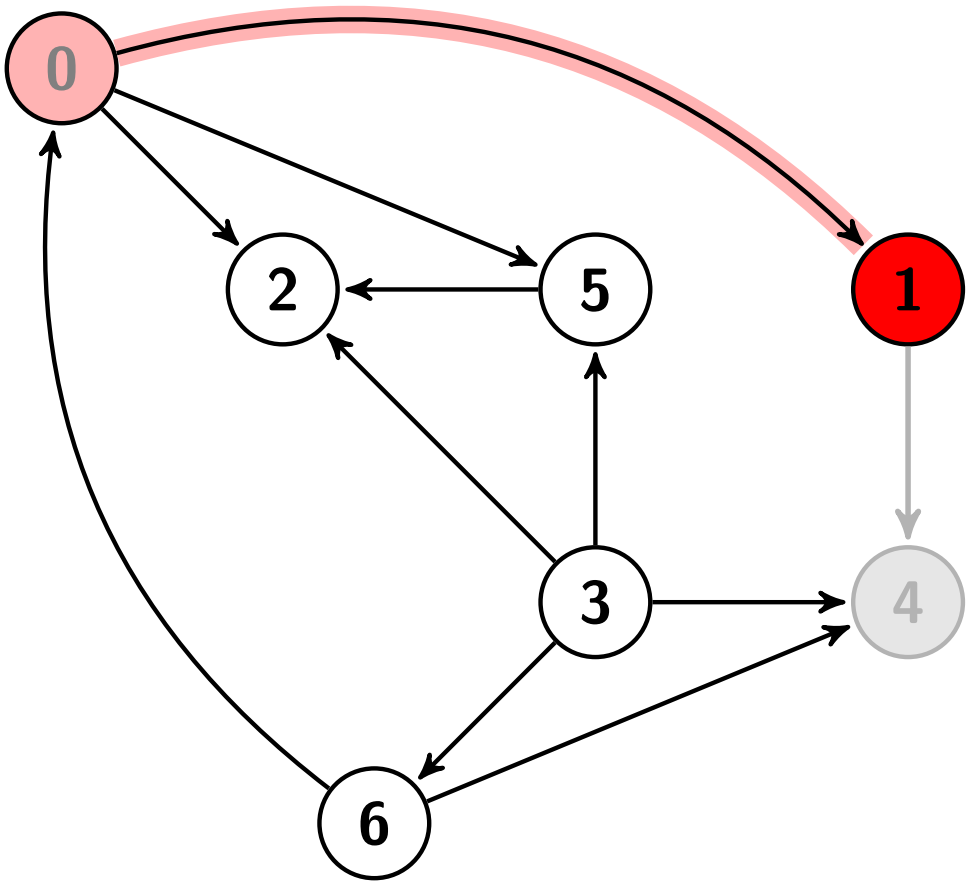
Pós-ordem - pilha

- {**4**}

Ação

- vértice 4 terminou
- **volta na recursão**

Ordenação topológica



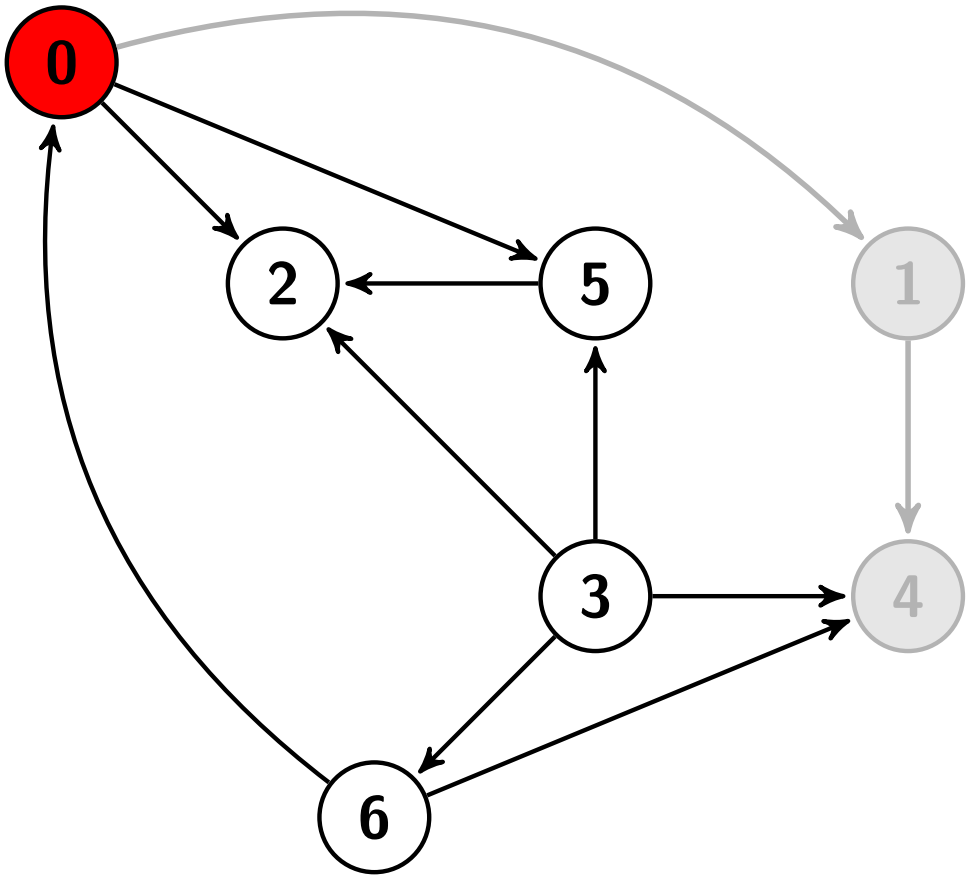
Pós-ordem - pilha

- $\{4, 1\}$

Ação

- vértice 1 terminou, põe na pilha

Ordenação topológica



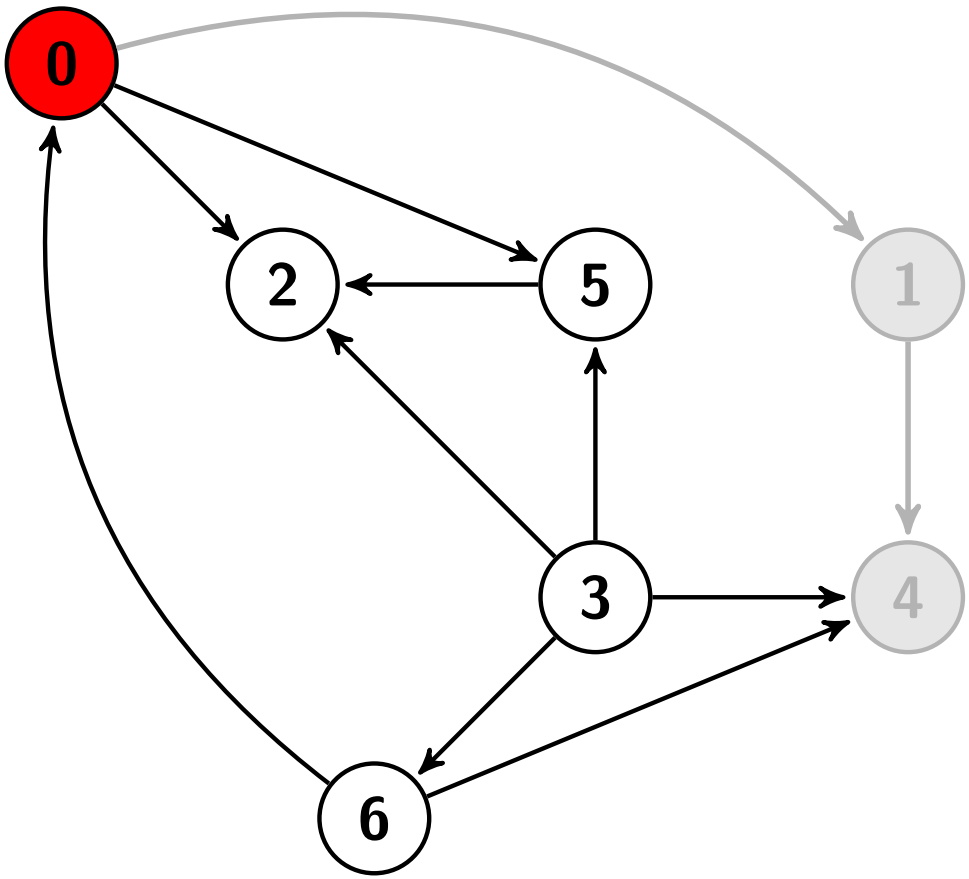
Pós-ordem - pilha

- $\{4, 1\}$

Ação

- vértice 1 terminou, põe na pilha
- **volta na recursão**

Ordenação topológica



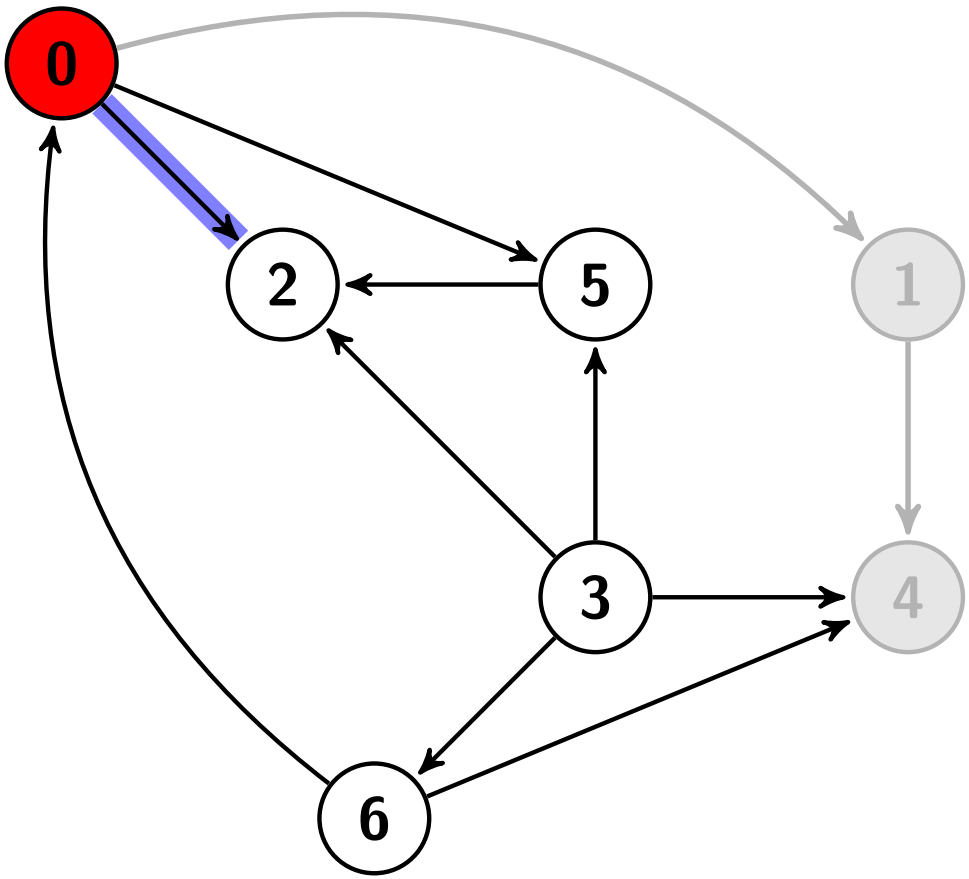
Pós-ordem - pilha

- $\{4, 1\}$

Ação

- visita 0; avaliar 1; avaliar 2; e avaliar 5

Ordenação topológica



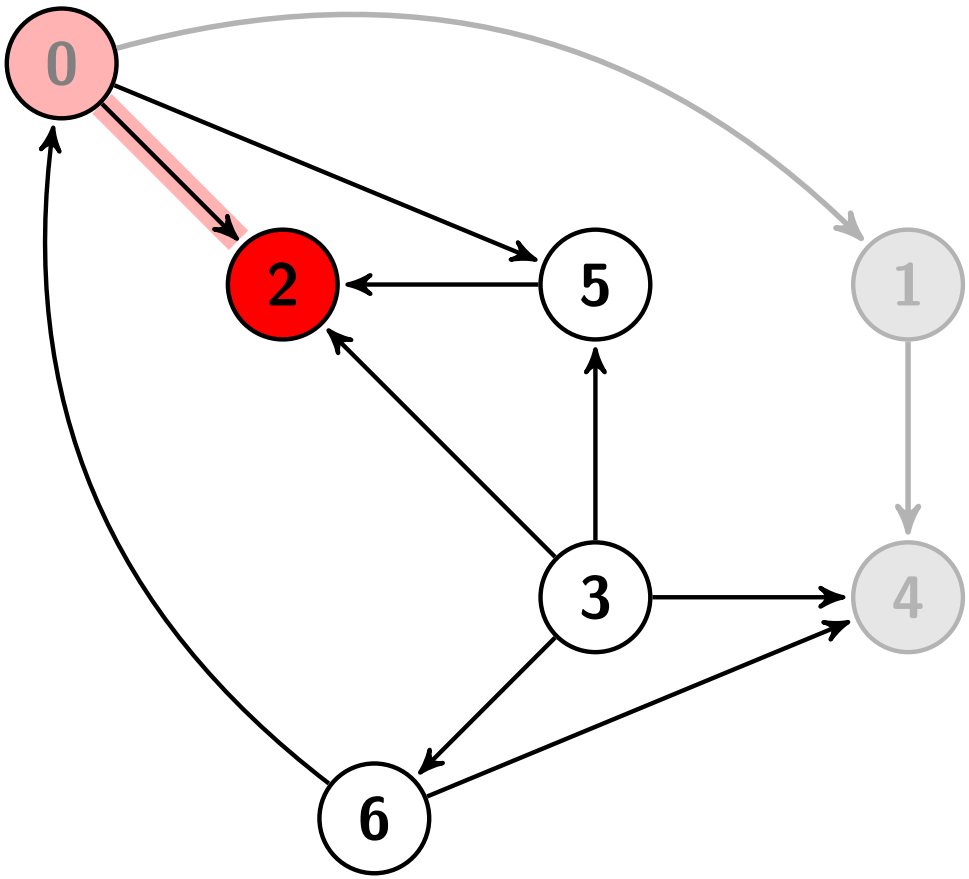
Pós-ordem - pilha

- $\{4, 1\}$

Ação

- visita 0; avaliar 1; avaliar 2; e avaliar 5

Ordenação topológica



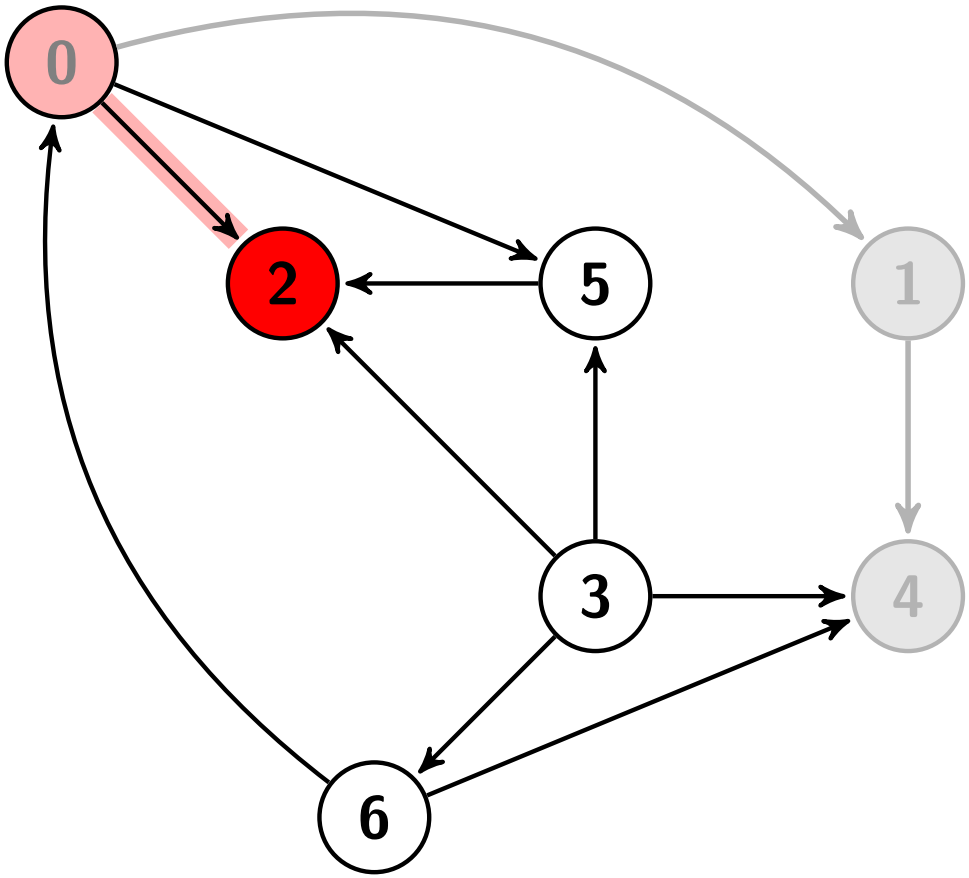
Pós-ordem - pilha

- $\{4, 1\}$

Ação

- visita 2;

Ordenação topológica



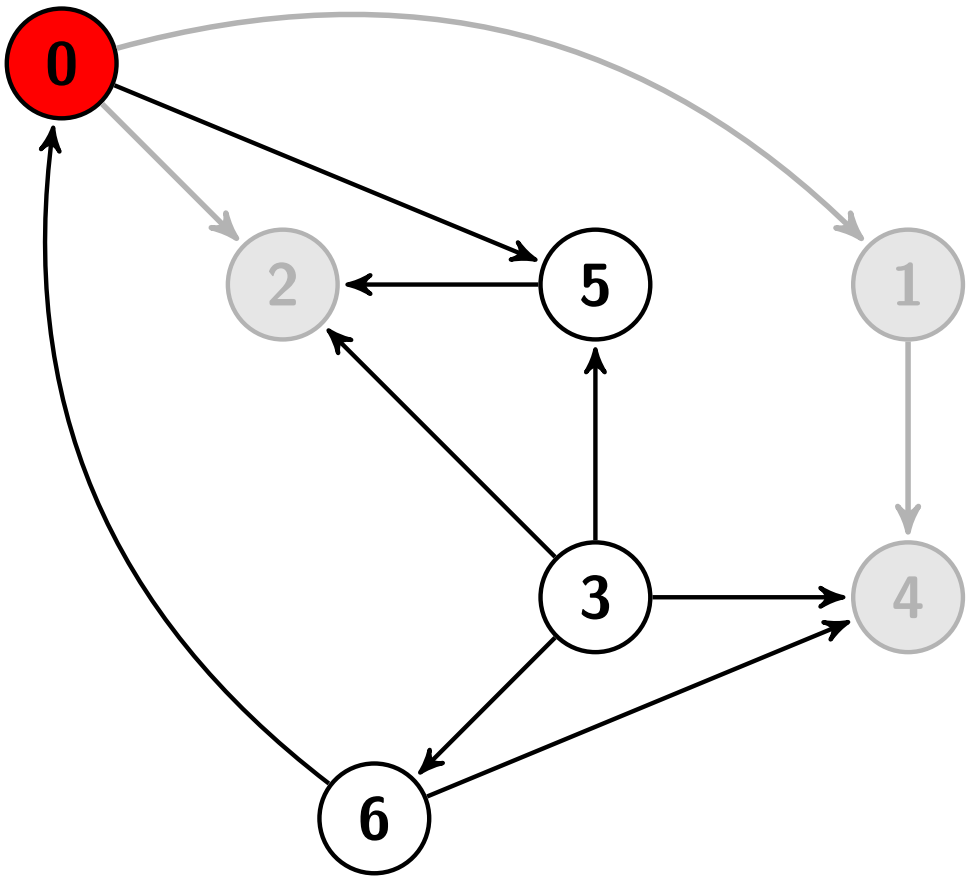
Pós-ordem - pilha

- $\{4, 1, 2\}$

Ação

- visita 2;
- Terminou o **2**. Empilha.

Ordenação topológica



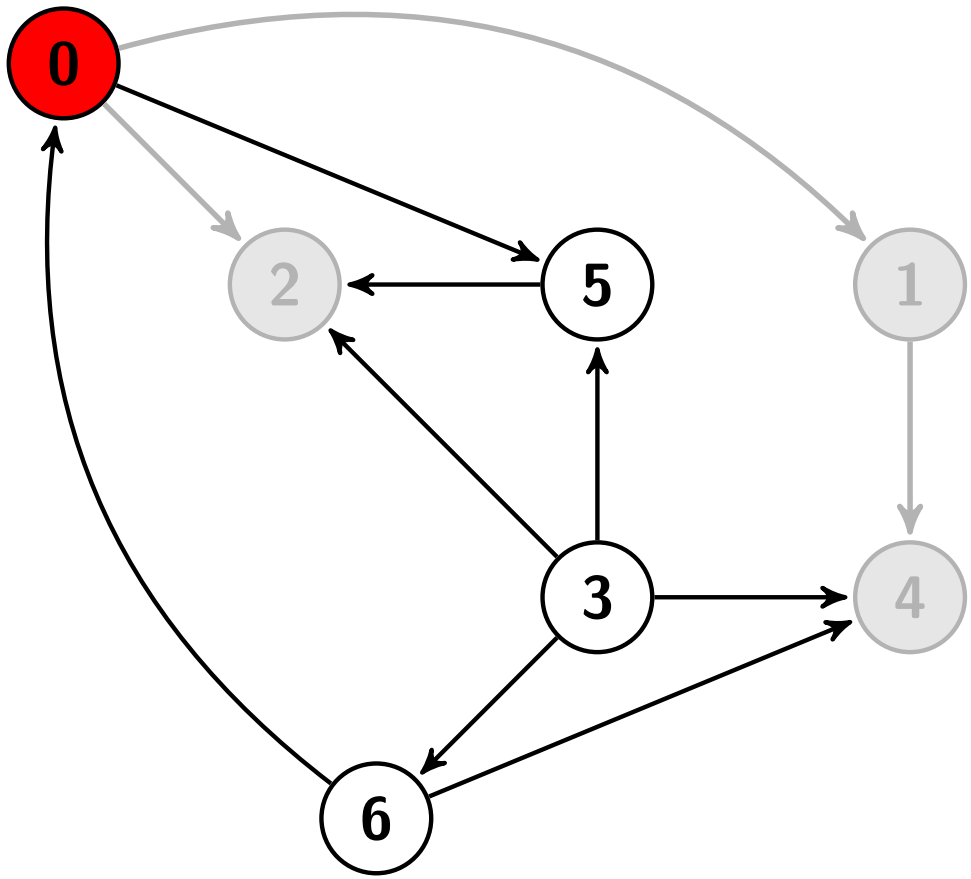
Pós-ordem - pilha

- $\{4, 1, 2\}$

Ação

- Retorna na recursão.

Ordenação topológica



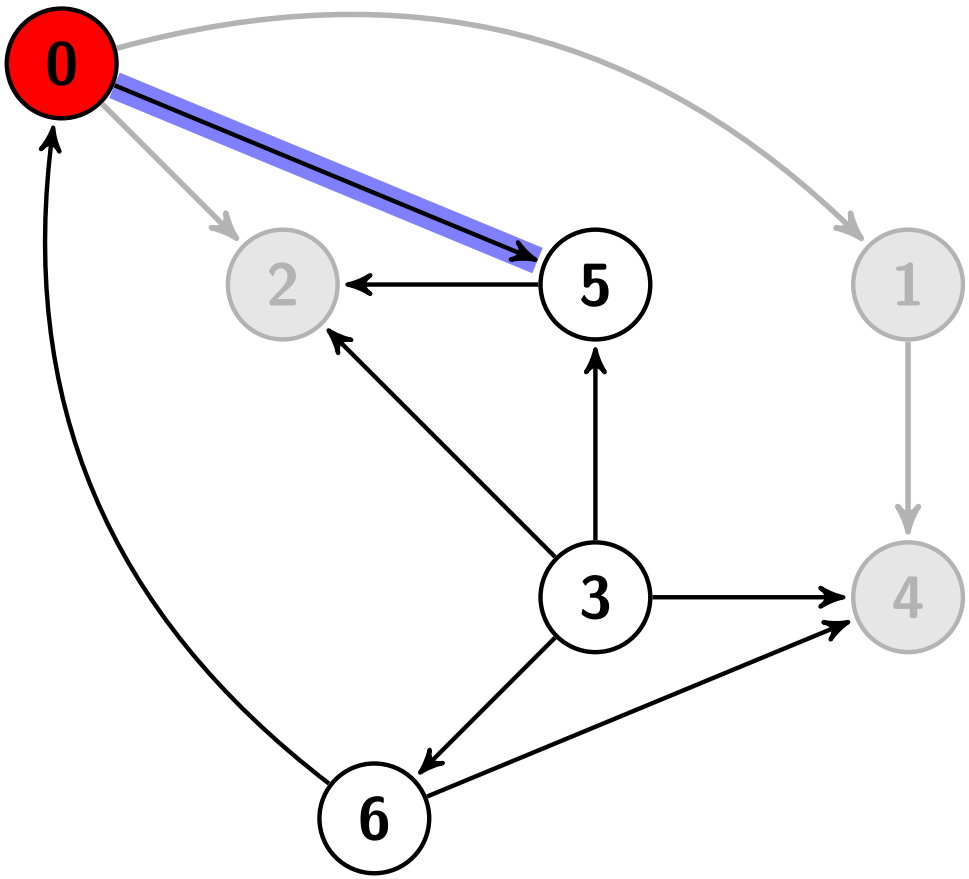
Pós-ordem - pilha

- $\{4, 1, 2\}$

Ação

- visita 0; avaliar 1; avaliar 2; e avaliar 5

Ordenação topológica



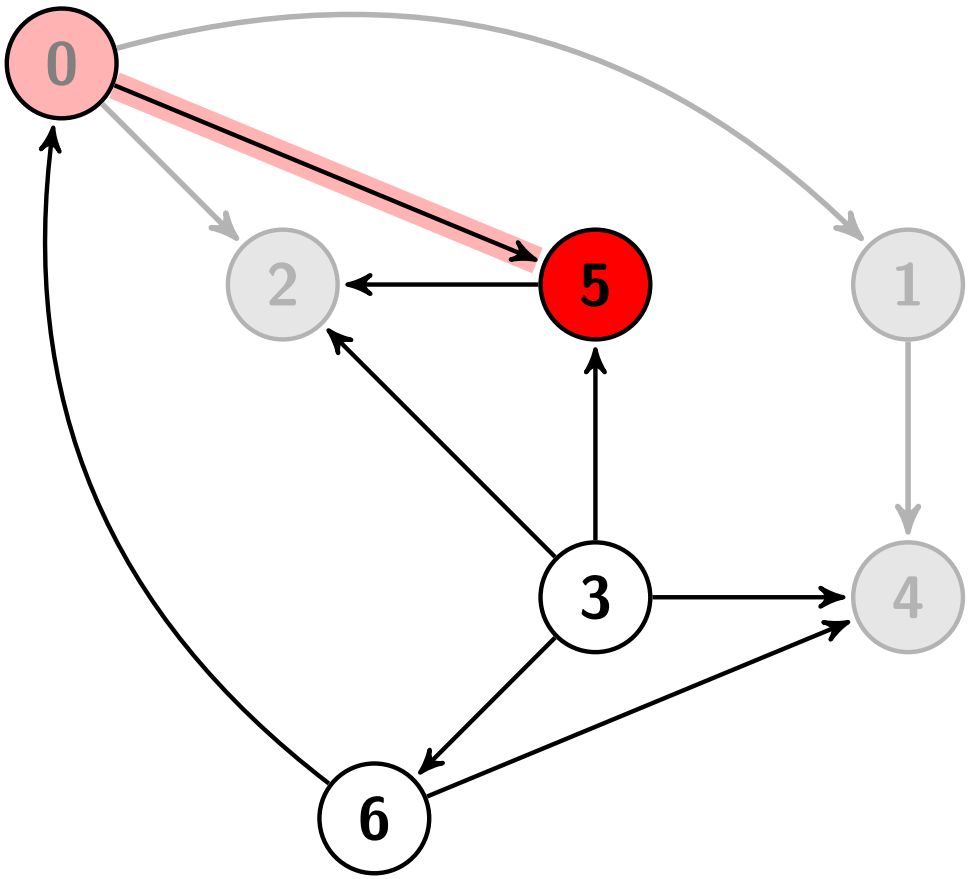
Pós-ordem - pilha

- $\{4, 1, 2\}$

Ação

- visita 0; avaliar 1; avaliar 2; e avaliar **5**

Ordenação topológica



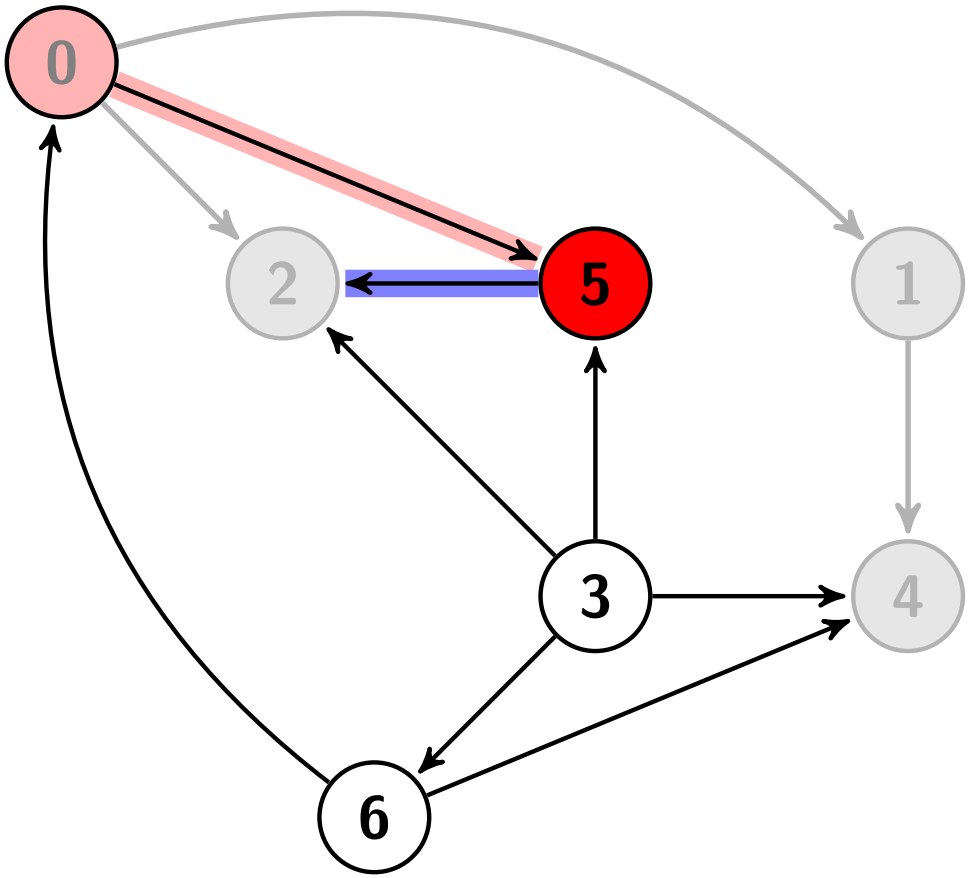
Pós-ordem - pilha

- $\{4, 1, 2\}$

Ação

- visita 5; avaliar 2

Ordenação topológica



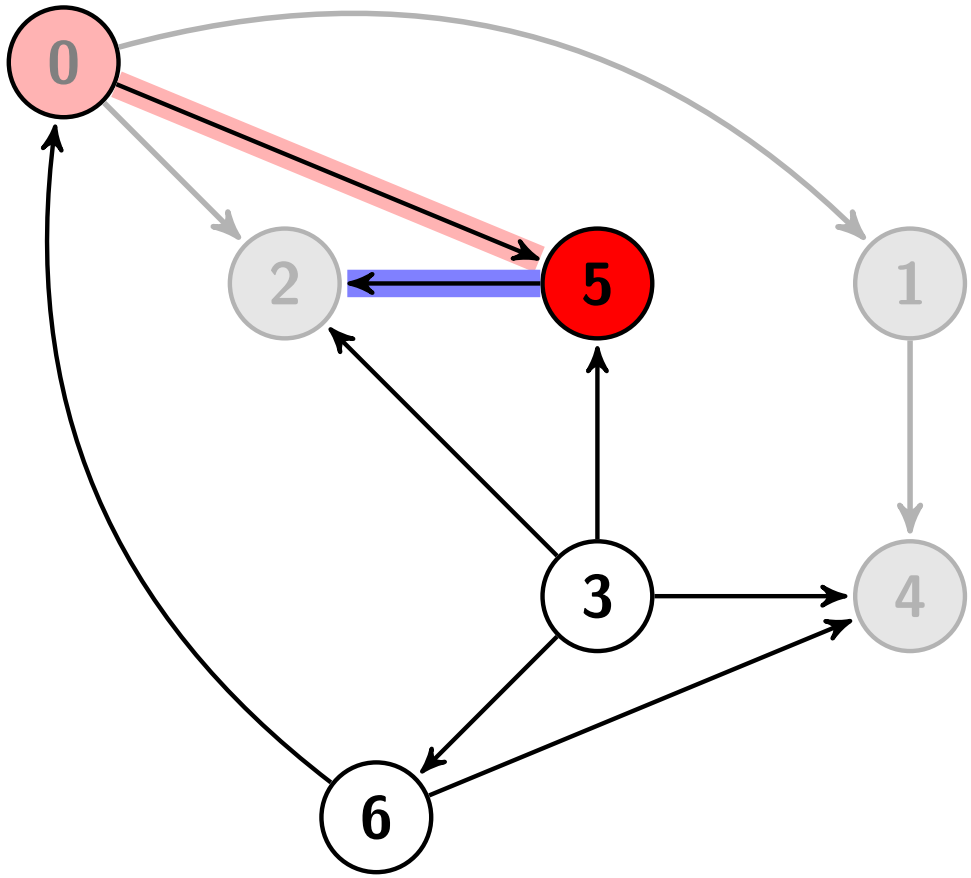
Pós-ordem - pilha

- $\{4, 1, 2\}$

Ação

- visita 5; avaliar **2**

Ordenação topológica



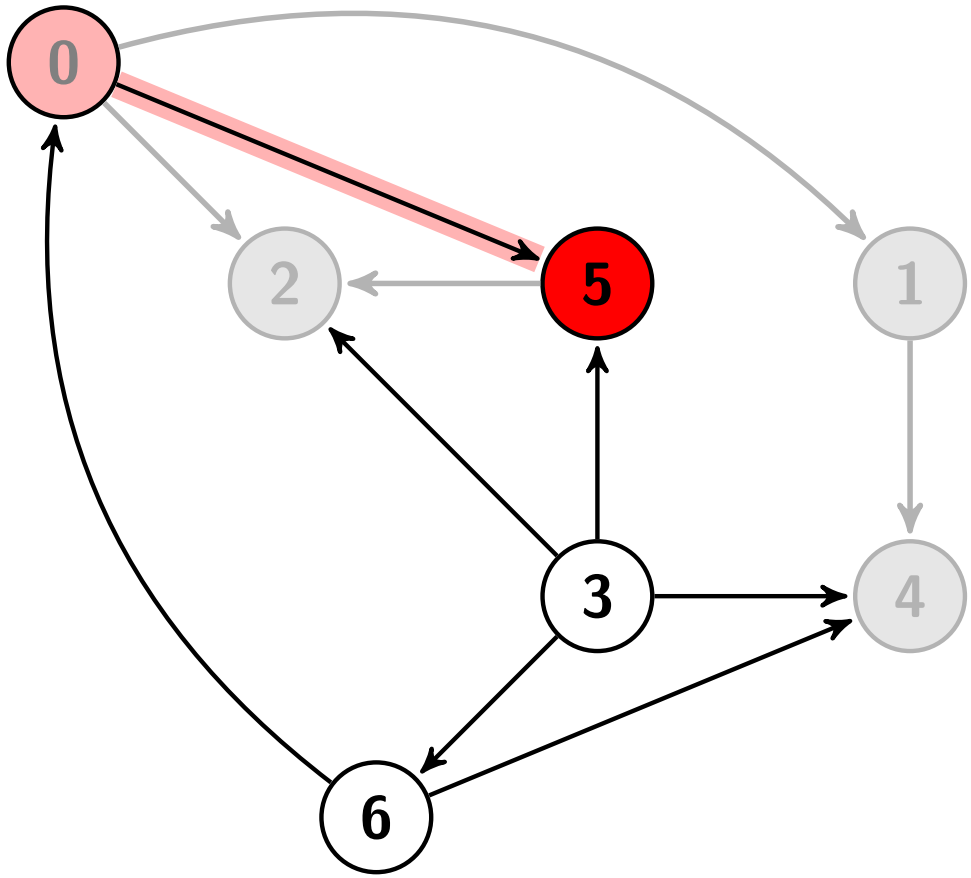
Pós-ordem - pilha

- $\{4, 1, 2\}$

Ação

- visita 5; avaliar **2**
- **2 já está marcado**

Ordenação topológica



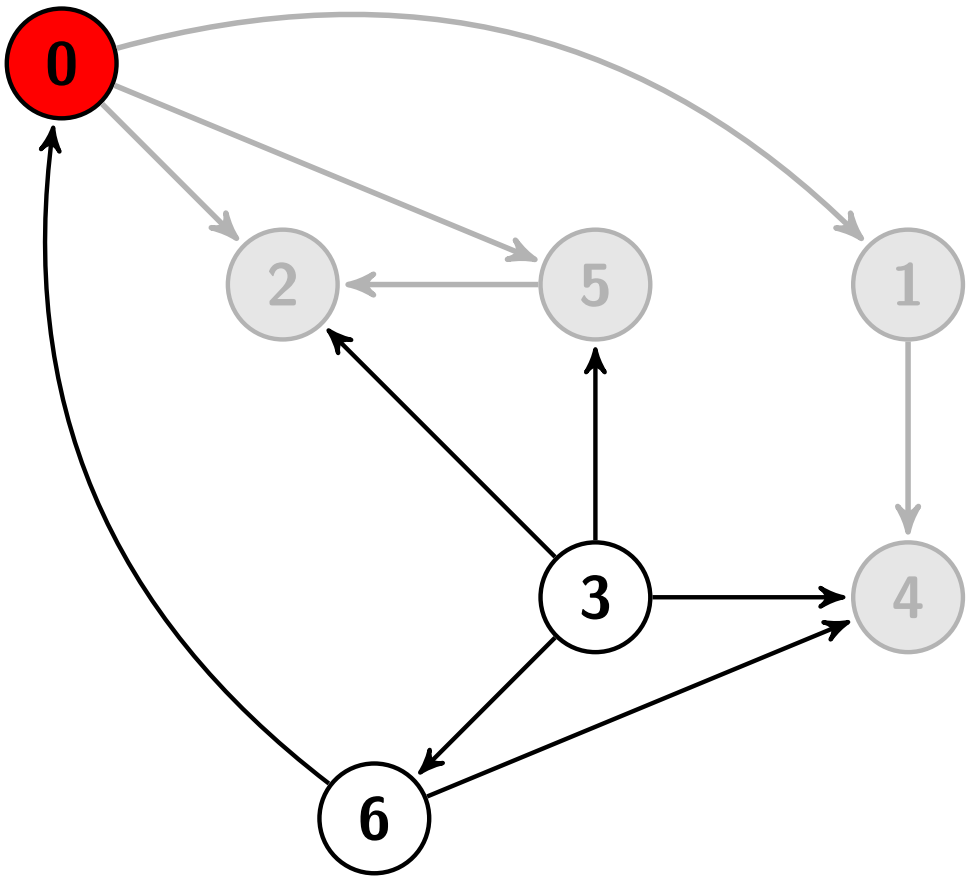
Pós-ordem - pilha

- $\{4, 1, 2, 5\}$

Ação

- 2 já está marcado.
- Retorna recursão. Terminou o **5**. Empilha.

Ordenação topológica



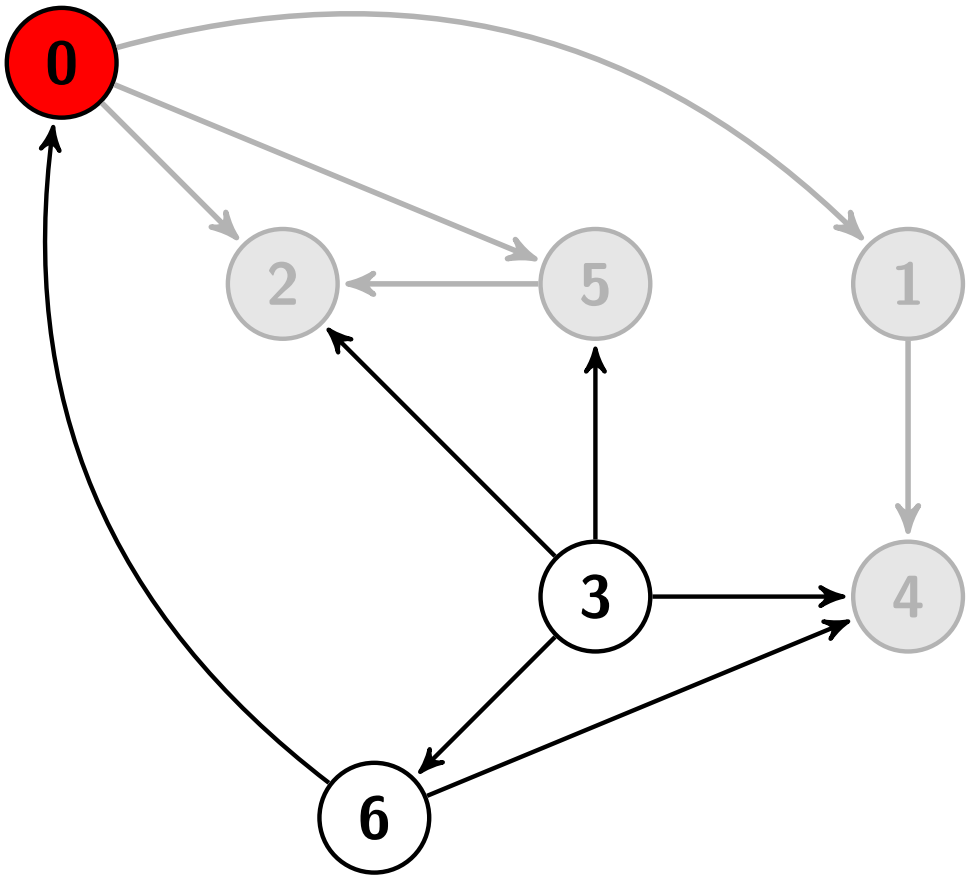
Pós-ordem - pilha

- $\{\mathbf{4, 1, 2, 5}\}$

Ação

- Terminou o 5. Retorna recursão.

Ordenação topológica



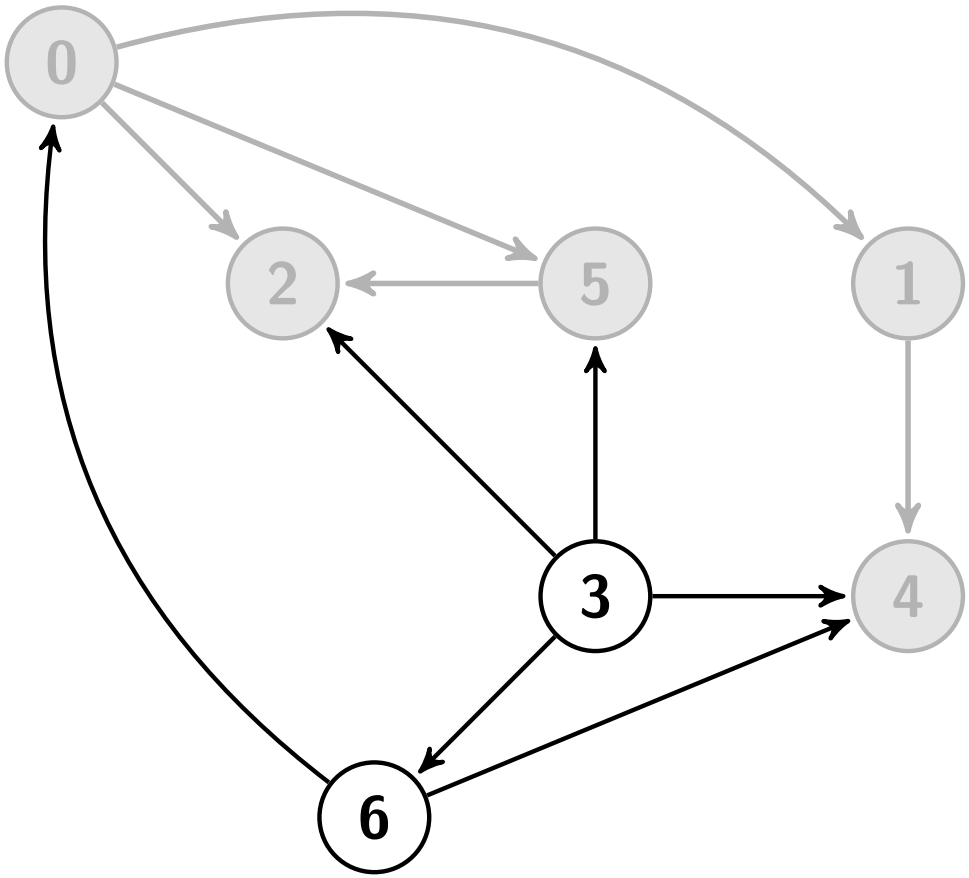
Pós-ordem - pilha

- $\{\mathbf{4, 1, 2, 5, 0}\}$

Ação

- Terminou o **0**. Empilha.

Ordenação topológica



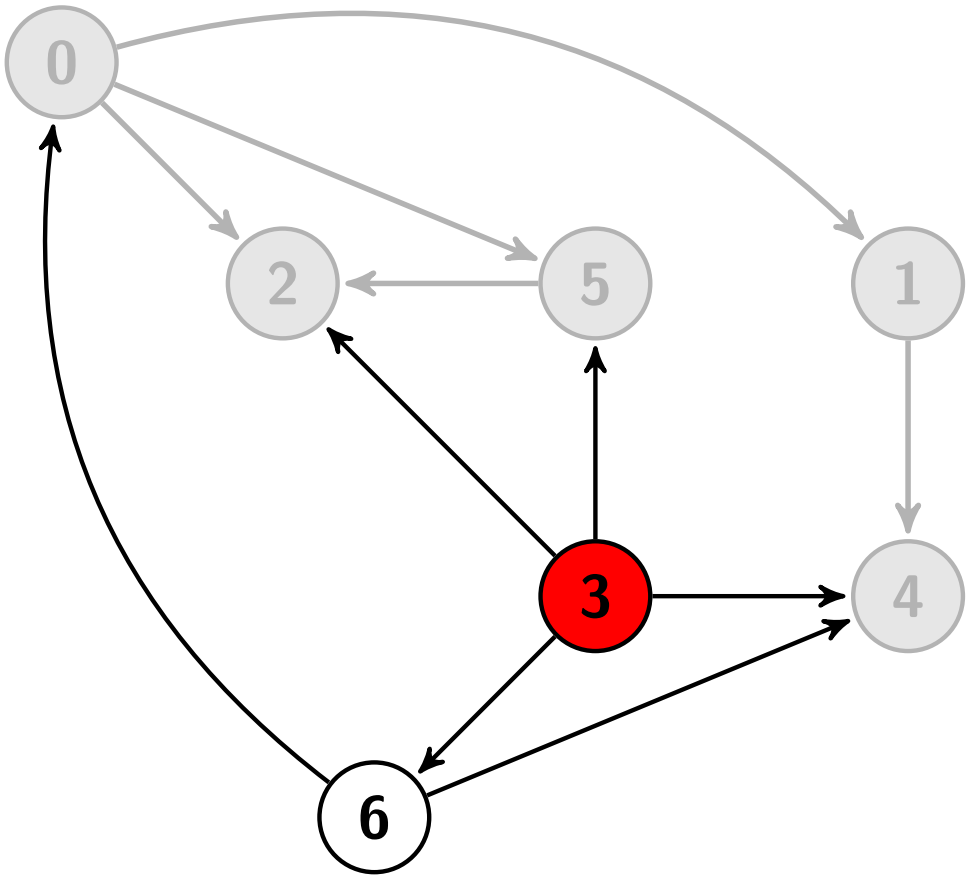
Pós-ordem - pilha

- $\{\mathbf{4, 1, 2, 5, 0}\}$

Ação

- Terminou a pós-ordem a partir de **0**
- Ainda temos **vértices não marcados**.

Ordenação topológica



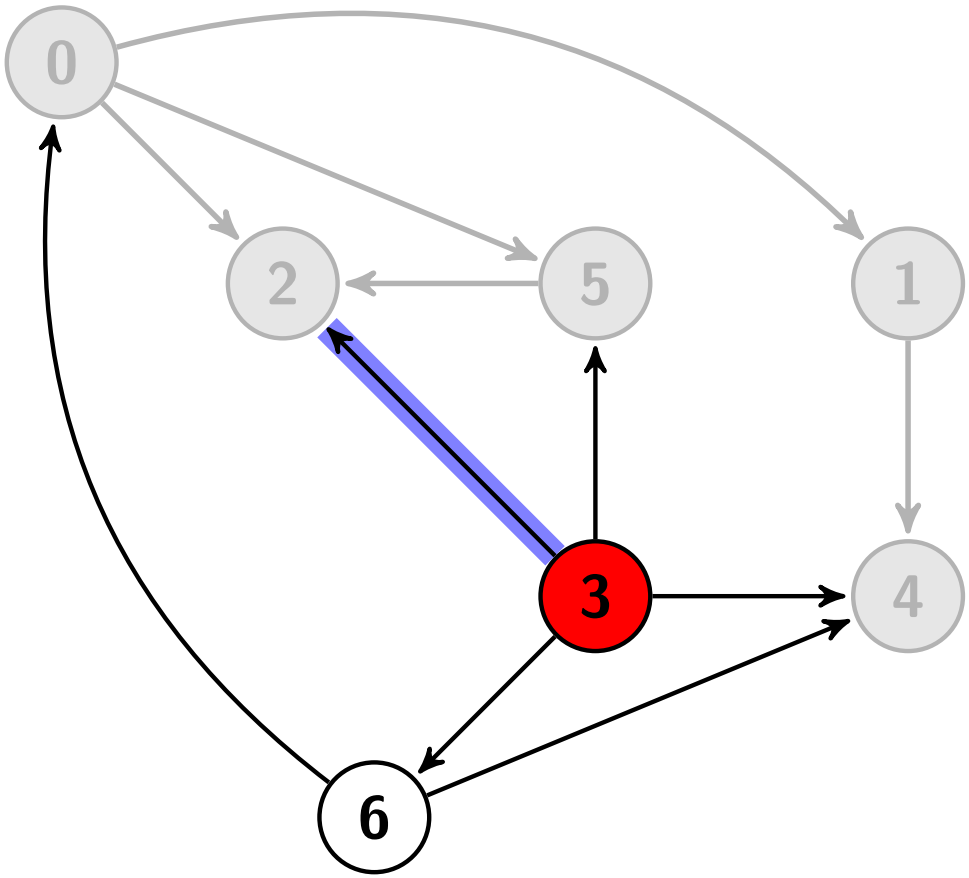
Pós-ordem - pilha

- $\{\mathbf{4, 1, 2, 5, 0}\}$

Ação

- Procurar próximo não marcado: **3**
- visita 3; avaliar 2; avaliar 4; avaliar 5; e avaliar 6.

Ordenação topológica



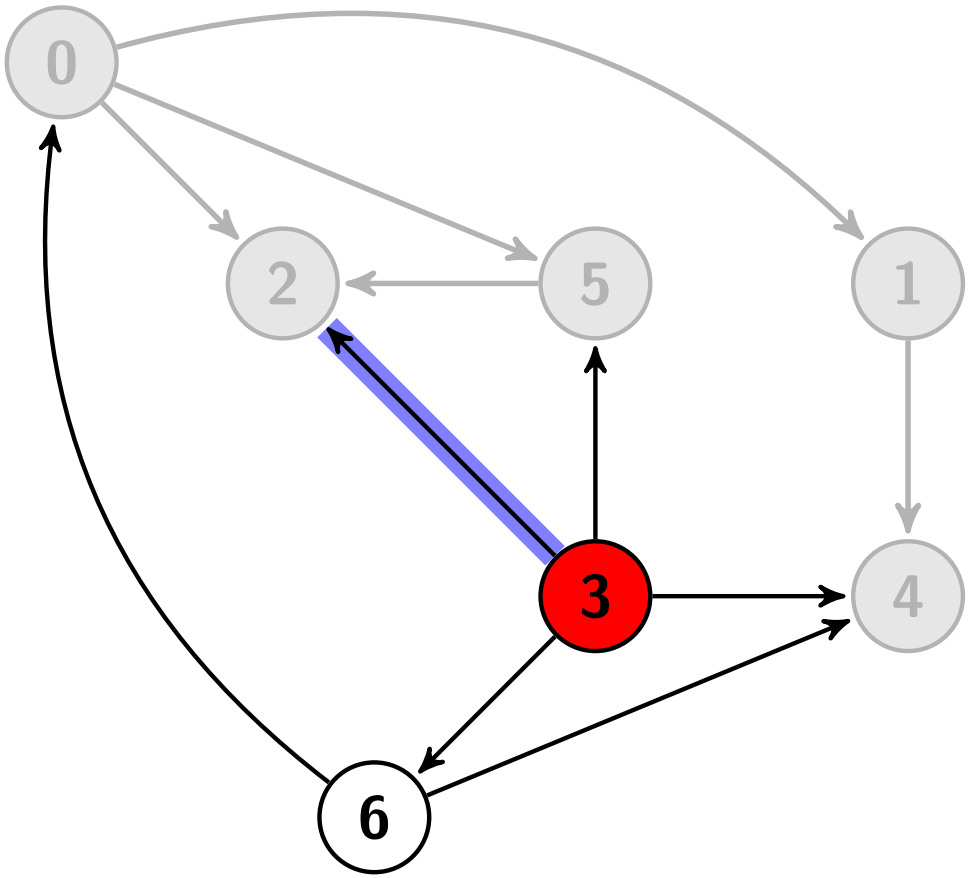
Pós-ordem - pilha

- $\{\mathbf{4, 1, 2, 5, 0}\}$

Ação

- visita 3; avaliar **2**; avaliar 4; avaliar 5; e avaliar 6.

Ordenação topológica



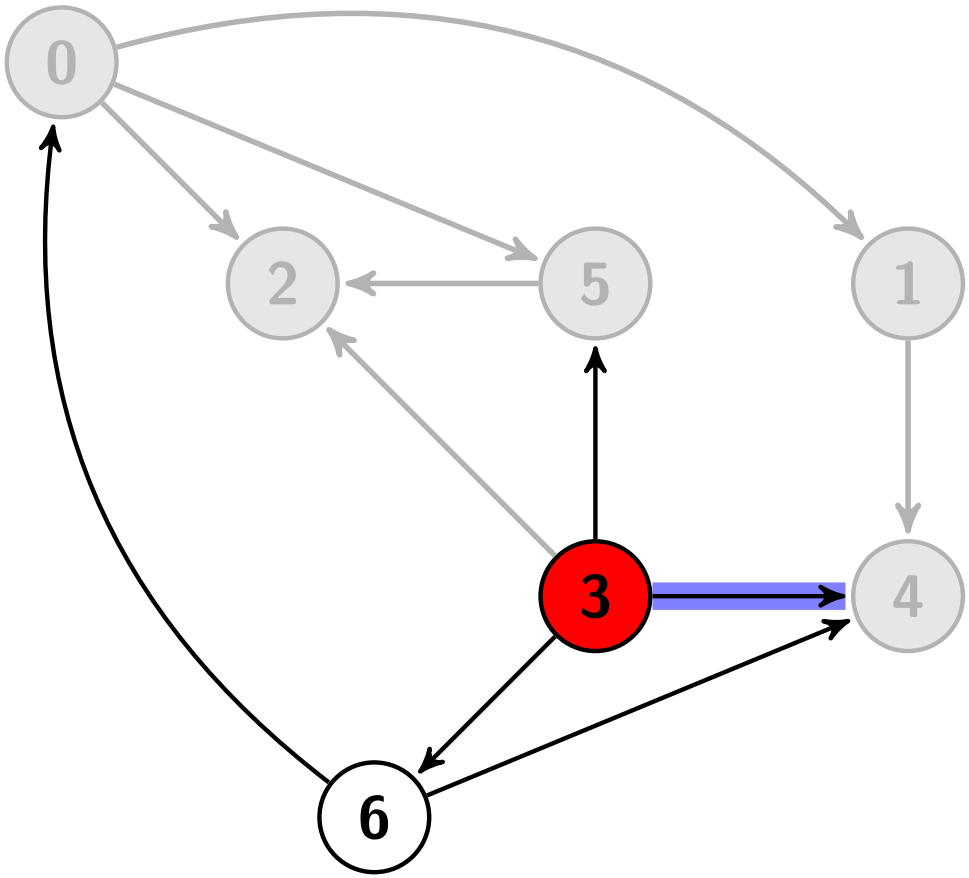
Pós-ordem - pilha

- $\{\mathbf{4, 1, 2, 5, 0}\}$

Ação

- visita 3; avaliar **2**; avaliar 4; avaliar 5; e avaliar 6.
- **2** está marcado.

Ordenação topológica



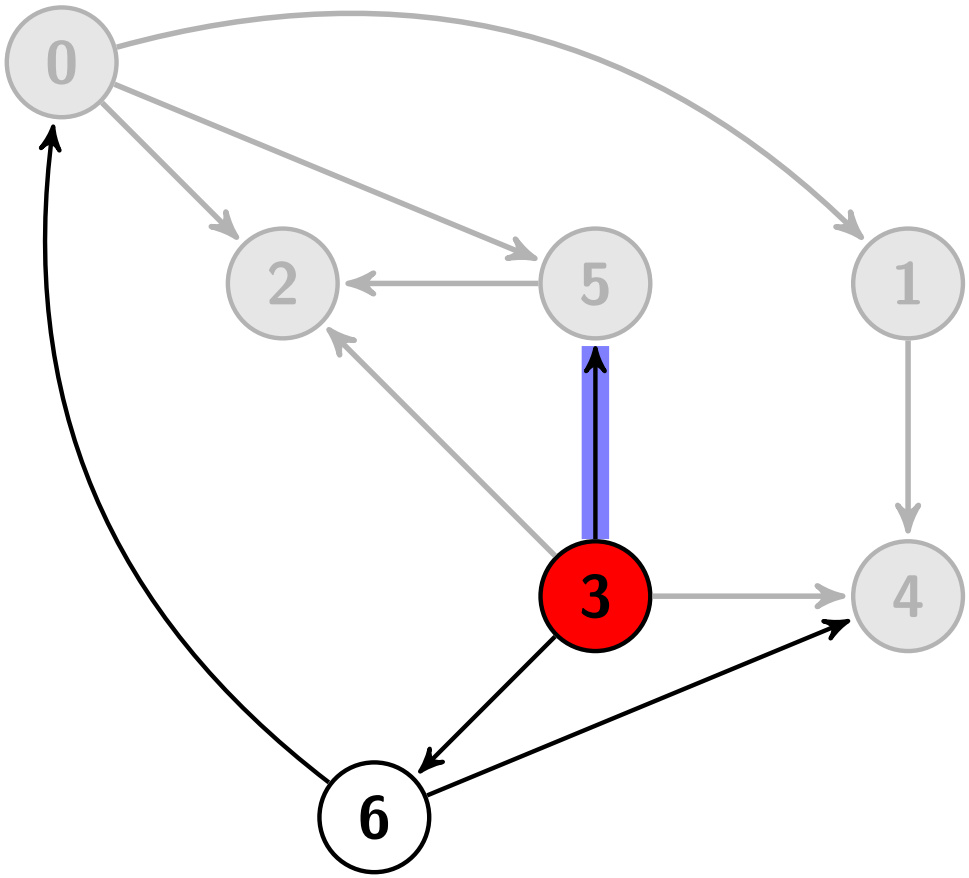
Pós-ordem - pilha

- $\{\mathbf{4, 1, 2, 5, 0}\}$

Ação

- visita 3; avaliar 2;avaliar **4**; avaliar 5; e avaliar 6.
- **4** está marcado.

Ordenação topológica



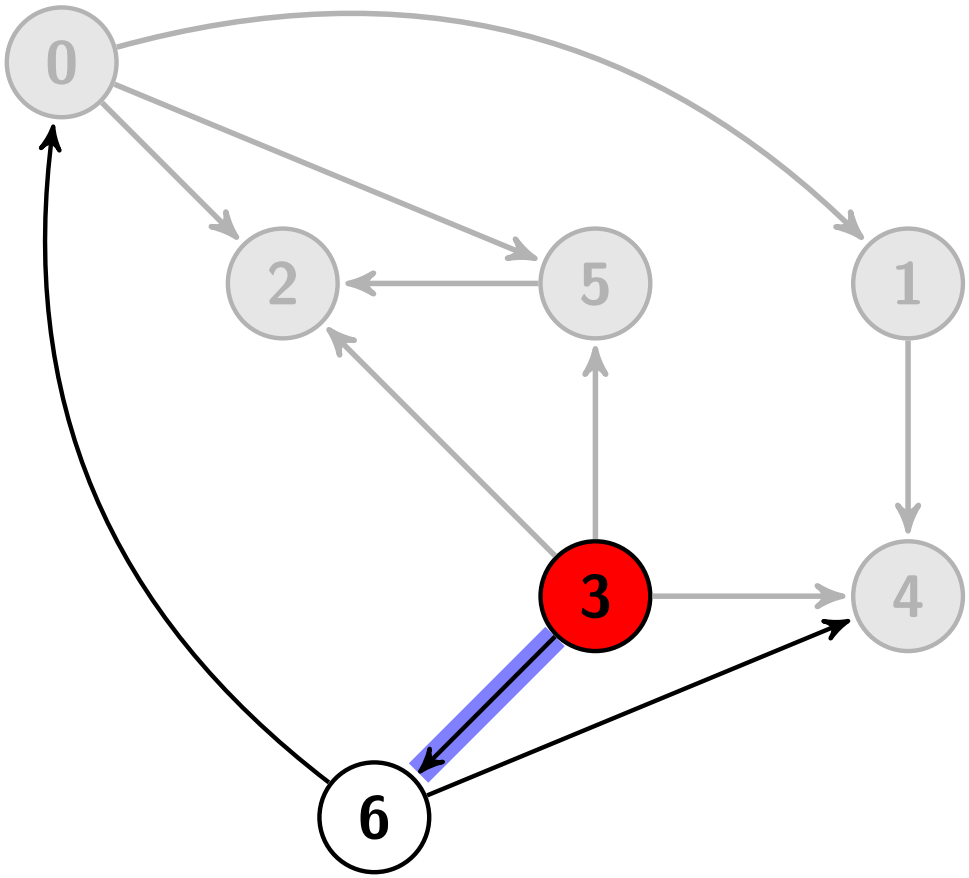
Pós-ordem - pilha

- $\{\mathbf{4, 1, 2, 5, 0}\}$

Ação

- visita 3; avaliar 2; avaliar 4; avaliar **5**; e avaliar 6.
- **5** está marcado.

Ordenação topológica



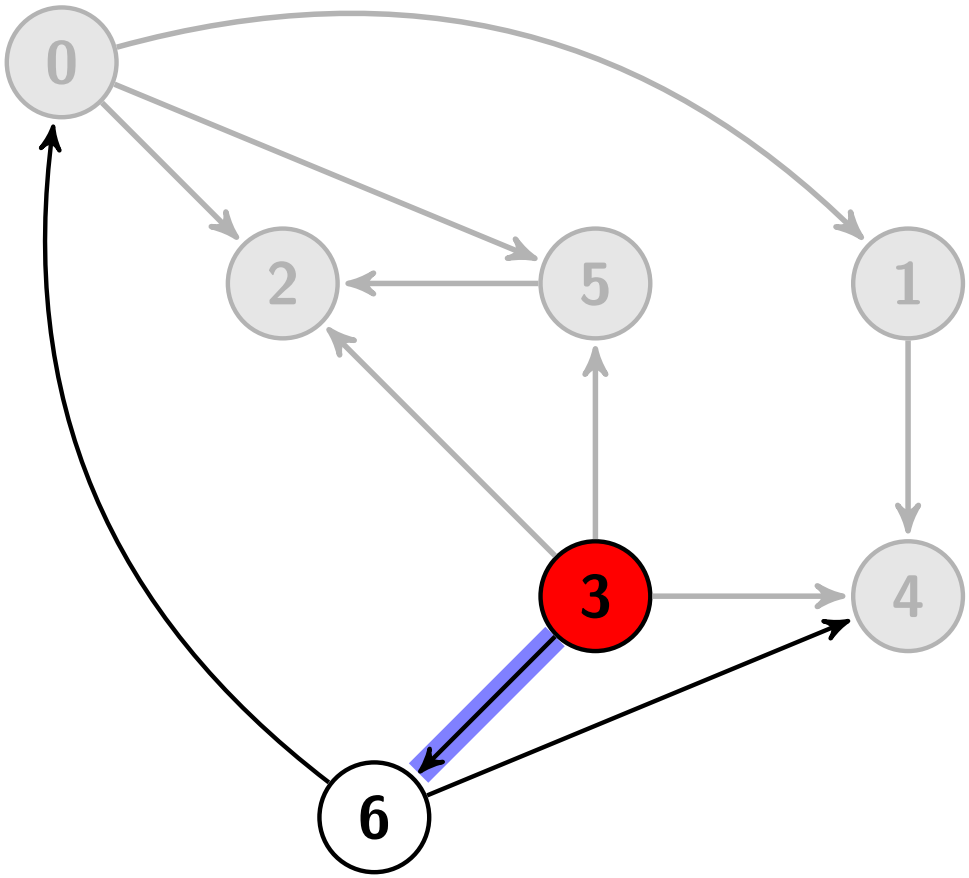
Pós-ordem - pilha

- $\{\mathbf{4, 1, 2, 5, 0}\}$

Ação

- visita 3; avaliar 2; avaliar 4; avaliar 5; e avaliar **6**.

Ordenação topológica



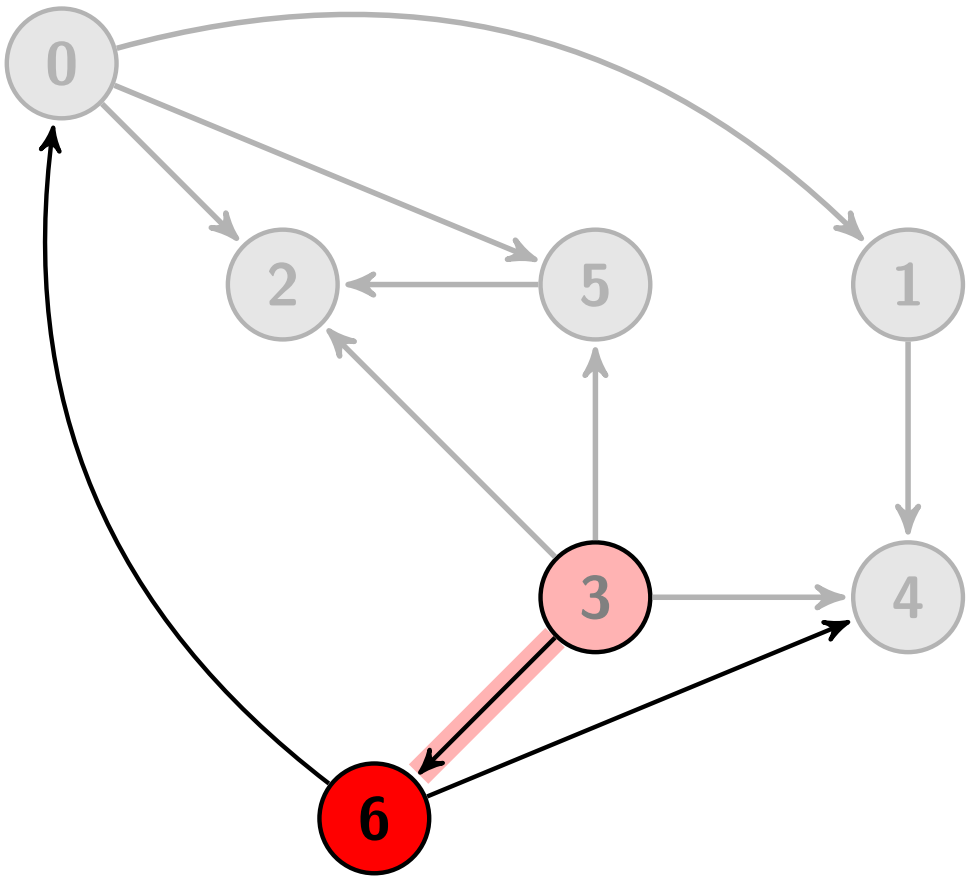
Pós-ordem - pilha

- $\{\mathbf{4, 1, 2, 5, 0}\}$

Ação

- visita 3; avaliar 2; avaliar 4; avaliar 5; e avaliar **6**.
- vértice **6 não** está marcado.

Ordenação topológica



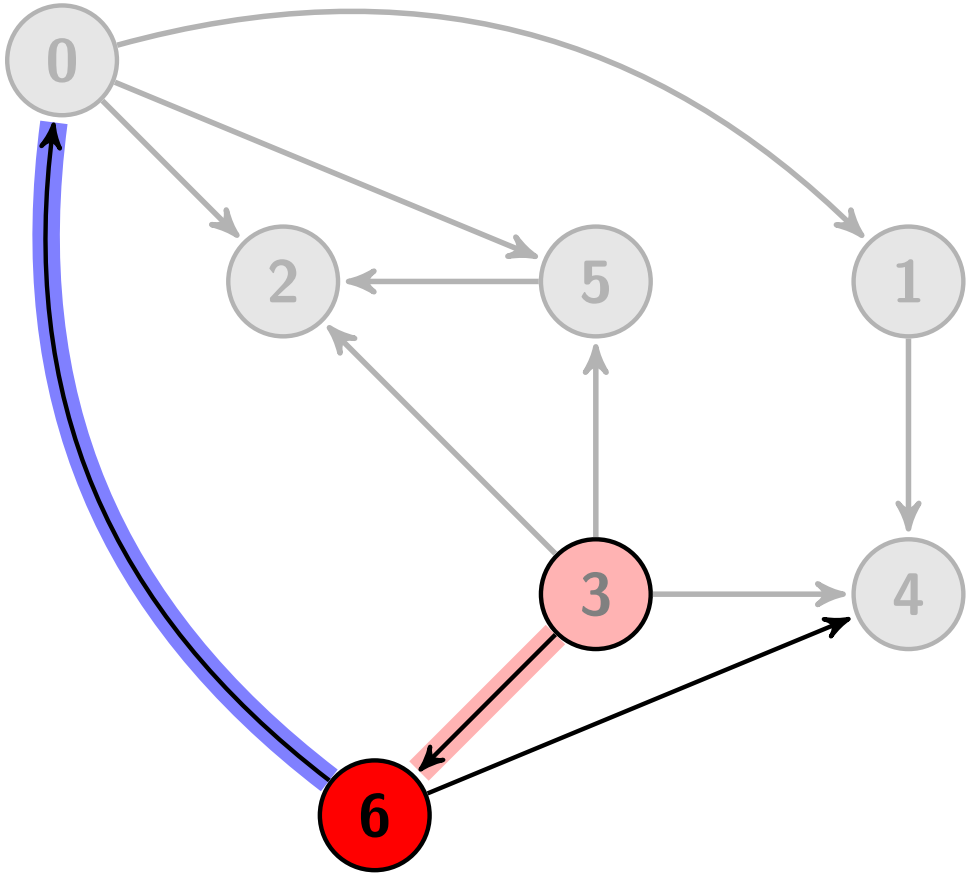
Pós-ordem - pilha

- $\{\mathbf{4, 1, 2, 5, 0}\}$

Ação

- visita 6; avaliar 0; e avaliar 4.

Ordenação topológica



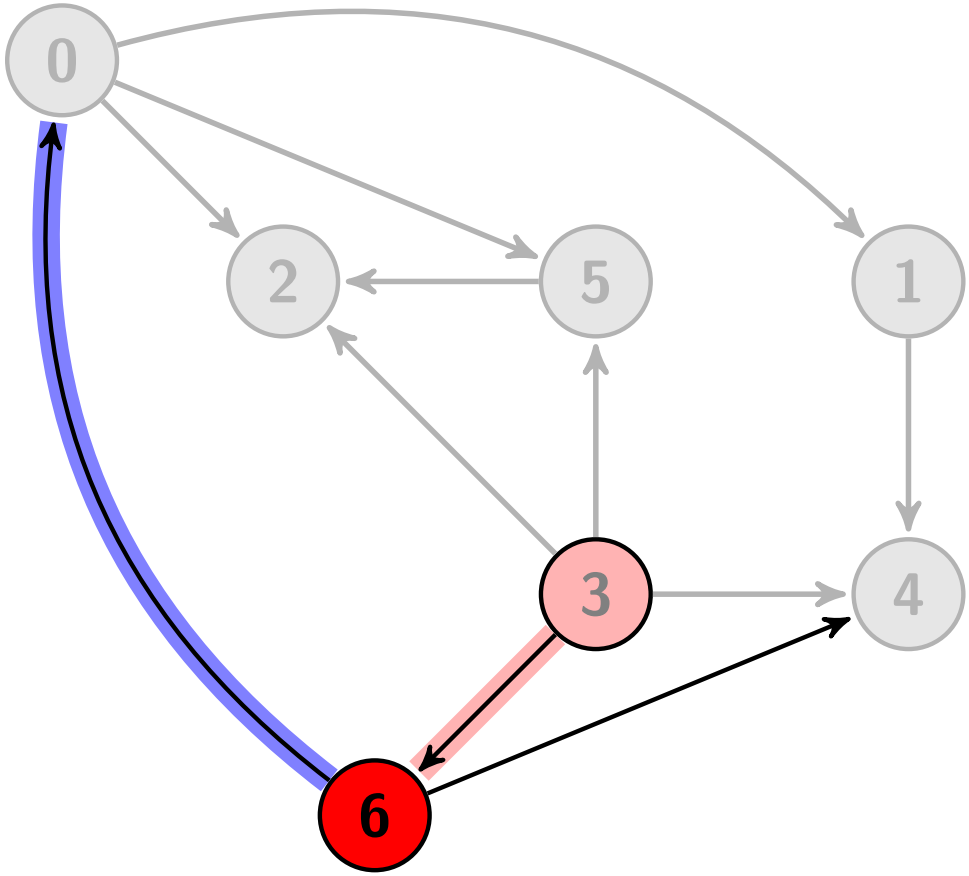
Pós-ordem - pilha

- $\{\mathbf{4, 1, 2, 5, 0}\}$

Ação

- visita 6; avaliar 0; e avaliar 4.

Ordenação topológica



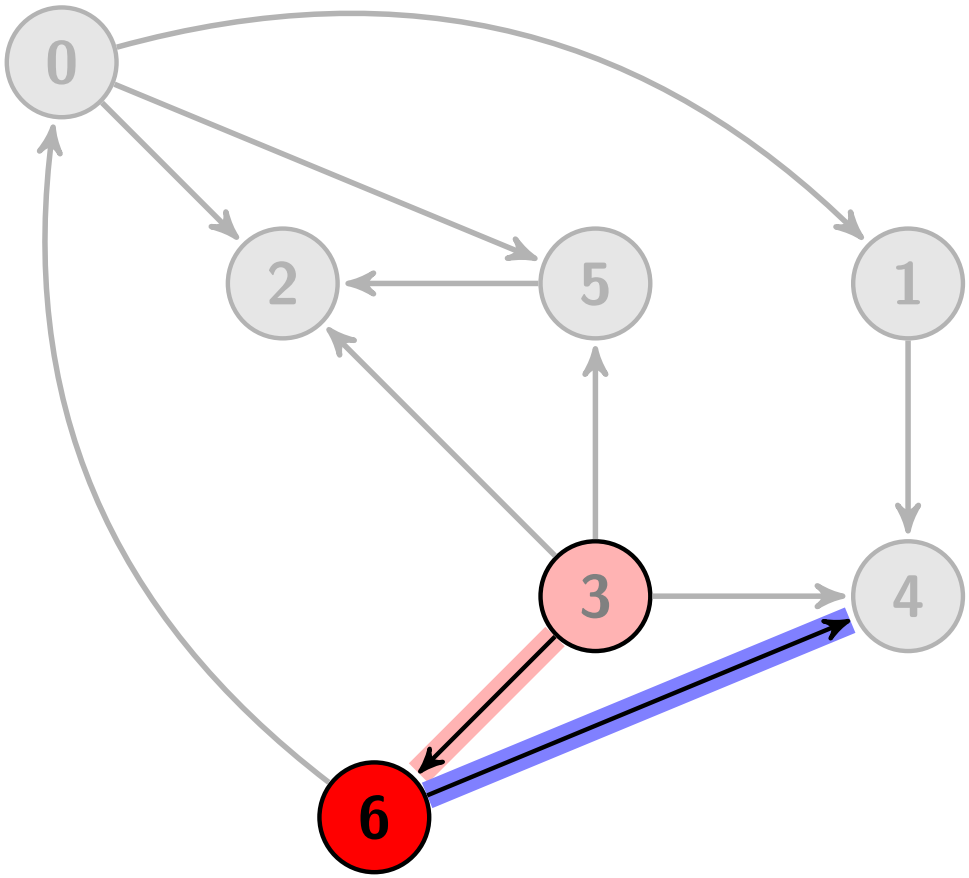
Pós-ordem - pilha

- $\{\mathbf{4, 1, 2, 5, 0}\}$

Ação

- visita 6; avaliar 0; e avaliar 4.
- vértice **0** está marcado.

Ordenação topológica



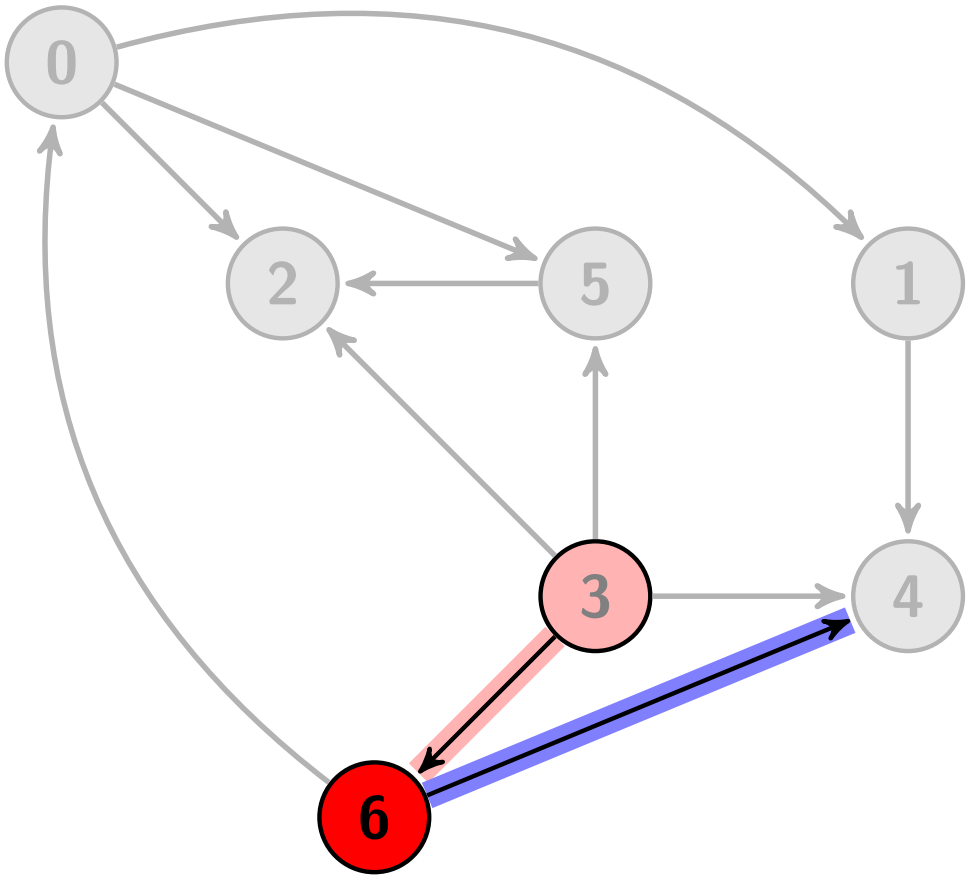
Pós-ordem - pilha

- $\{\mathbf{4, 1, 2, 5, 0}\}$

Ação

- visita 6; avaliar 0; e avaliar 4.

Ordenação topológica



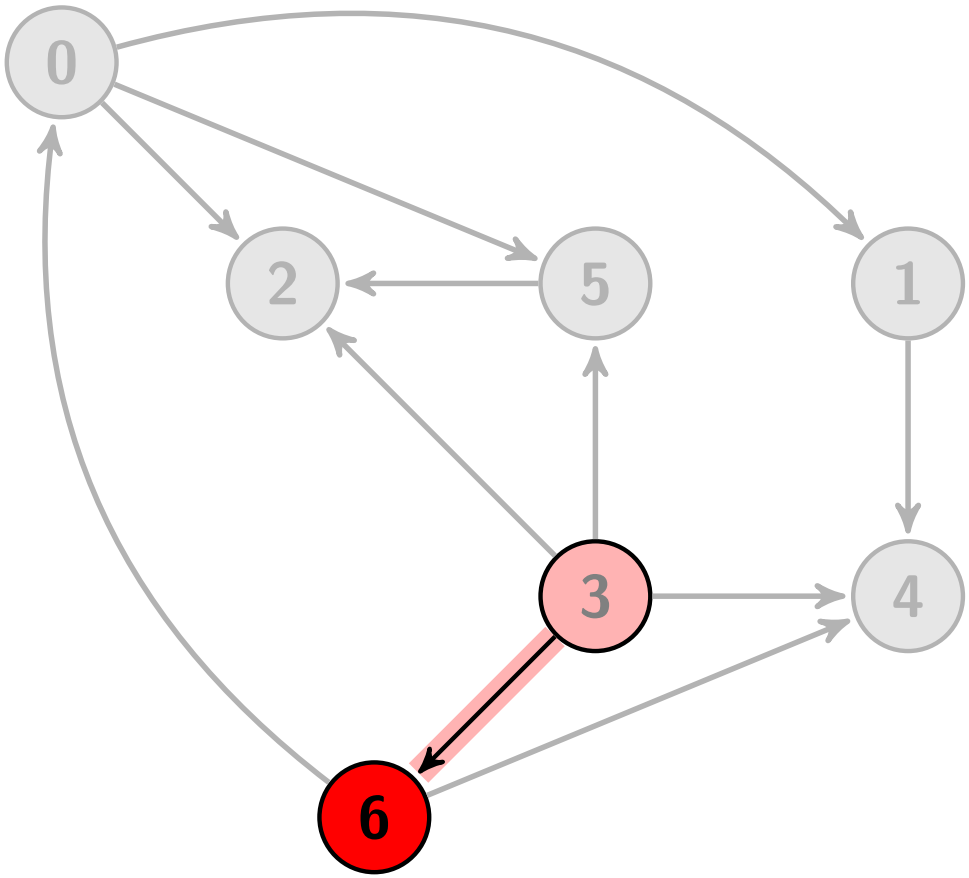
Pós-ordem - pilha

- $\{\mathbf{4, 1, 2, 5, 0}\}$

Ação

- visita 6; avaliar 0; e avaliar 4.
- vértice **4** está marcado.

Ordenação topológica



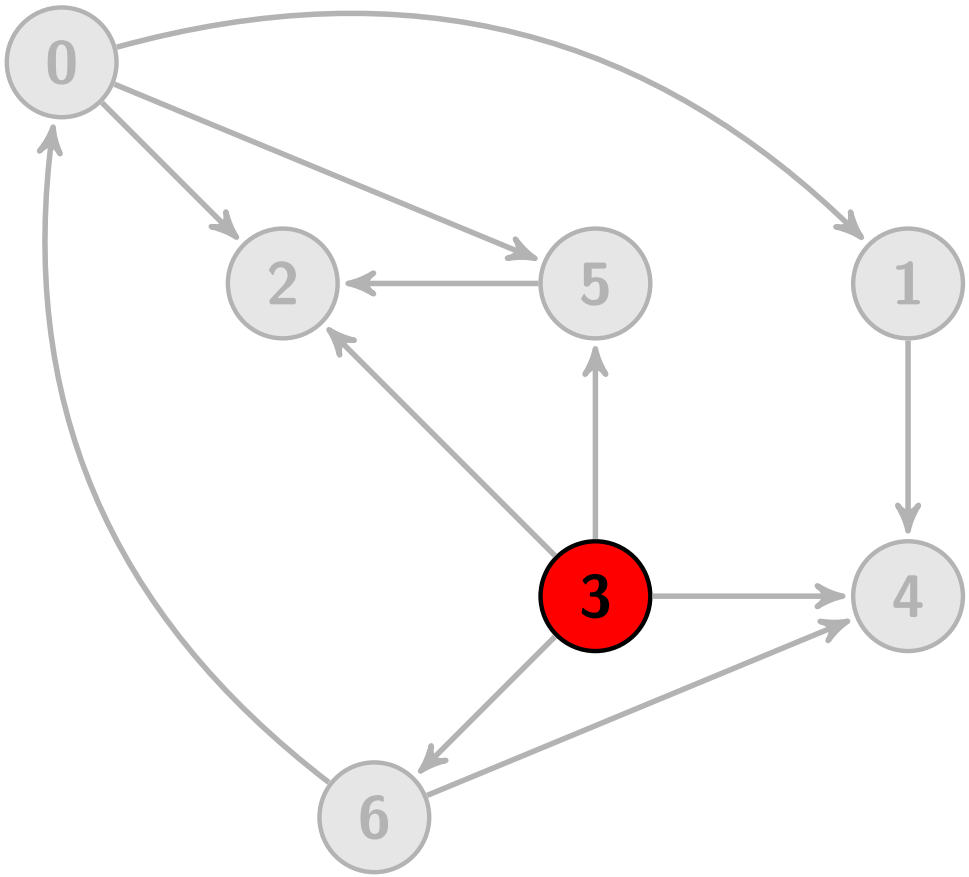
Pós-ordem - pilha

- $\{\mathbf{4, 1, 2, 5, 0, 6}\}$

Ação

- visita 6; avaliar 0; e avaliar 4.
- Terminou o **6**. Empilha.

Ordenação topológica



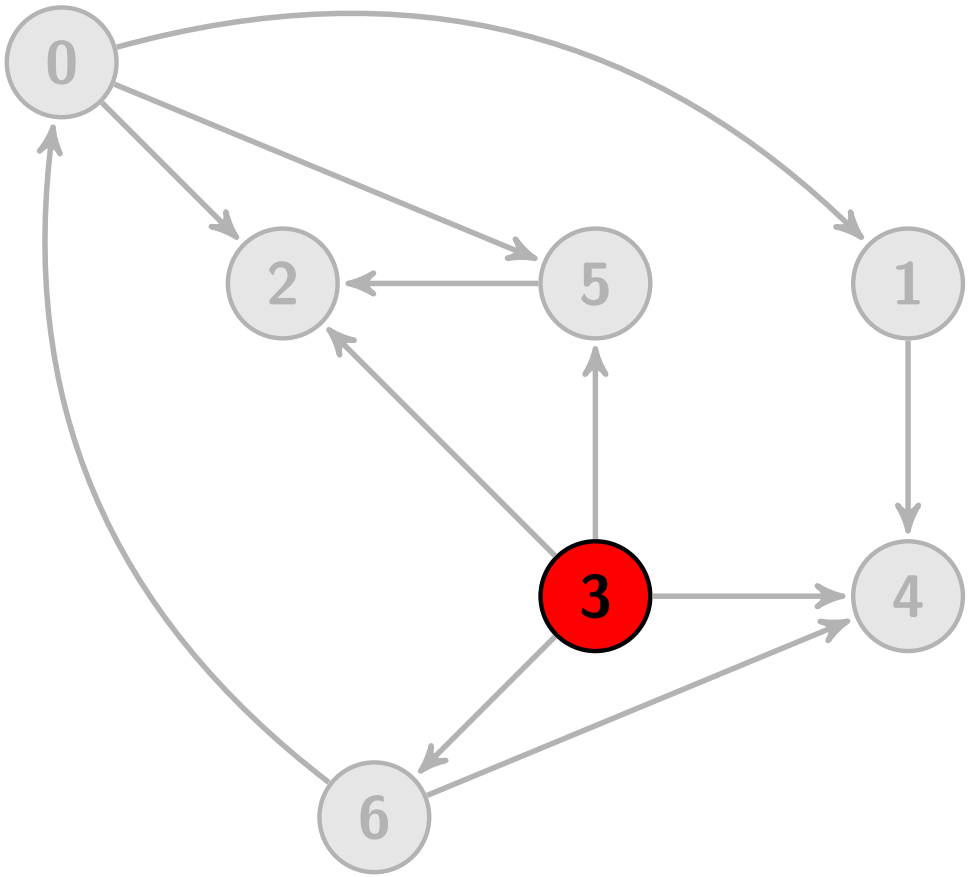
Pós-ordem - pilha

- $\{\mathbf{4, 1, 2, 5, 0, 6}\}$

Ação

- Terminou o 6.
- Retorna na recursão.

Ordenação topológica



Pós-ordem - pilha

- $\{\mathbf{4, 1, 2, 5, 0, 6, 3}\}$

Ação

- Terminou o **3**. Empilha.
- Retorna nas recursões e fim do algoritmo.

Menor caminho em dígrafos com pesos

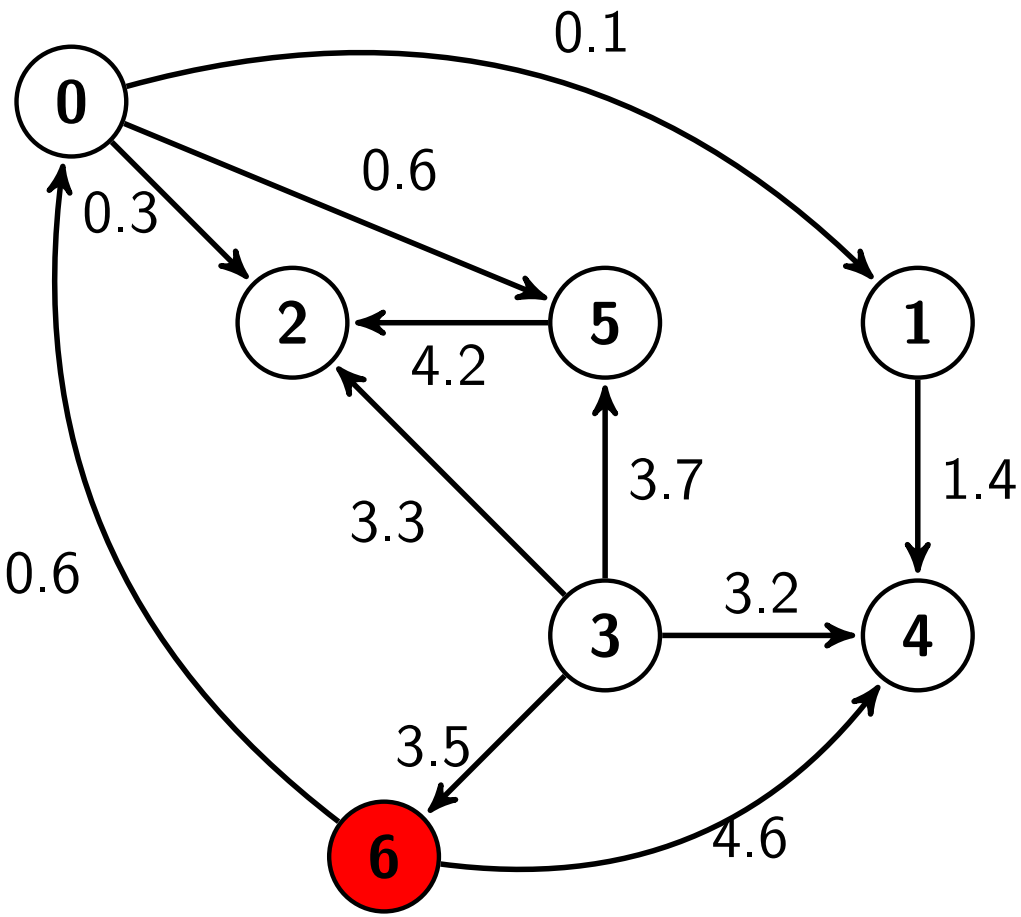
Existem diferentes modalidades

- Uma origem, todos os destinos
- Uma origem, um destino
- Todos os pares

Queremos o custo e o caminho.

```
1  class MenorCaminho {//em dígrafo acíclico c/ pesos
2   double[] distPara; Aresta[] arestaPara;
3
4   public MenorCaminho(Digrafo G, int origem) {
5    arestaPara = new Aresta[G.V()];
6    distPara = new double[G.V()];
7
8    for (int v = 0; v < G.V(); v++)
9      distPara[v] = Double.POSITIVE_INFINITY;
10   distPara[origem] = 0.0;
11
12   OrdemTopologica topo = new OrdemTopologica(G);
13   for (int v: topo.ordem())
14     for (Aresta a: G.adj(v)) expandir(a);
15  }
16
17  void expandir(Aresta a) {
18   int v = a.de(), w = a.para();
19   if (distPara[w] > distPara[v] + a.peso()) {
20     distPara[w] = distPara[v] + a.peso();
21     arestaPara[w] = a;
22   }
23  }
24 }
```

Menor caminho



Pós-ordem topológica - pilha

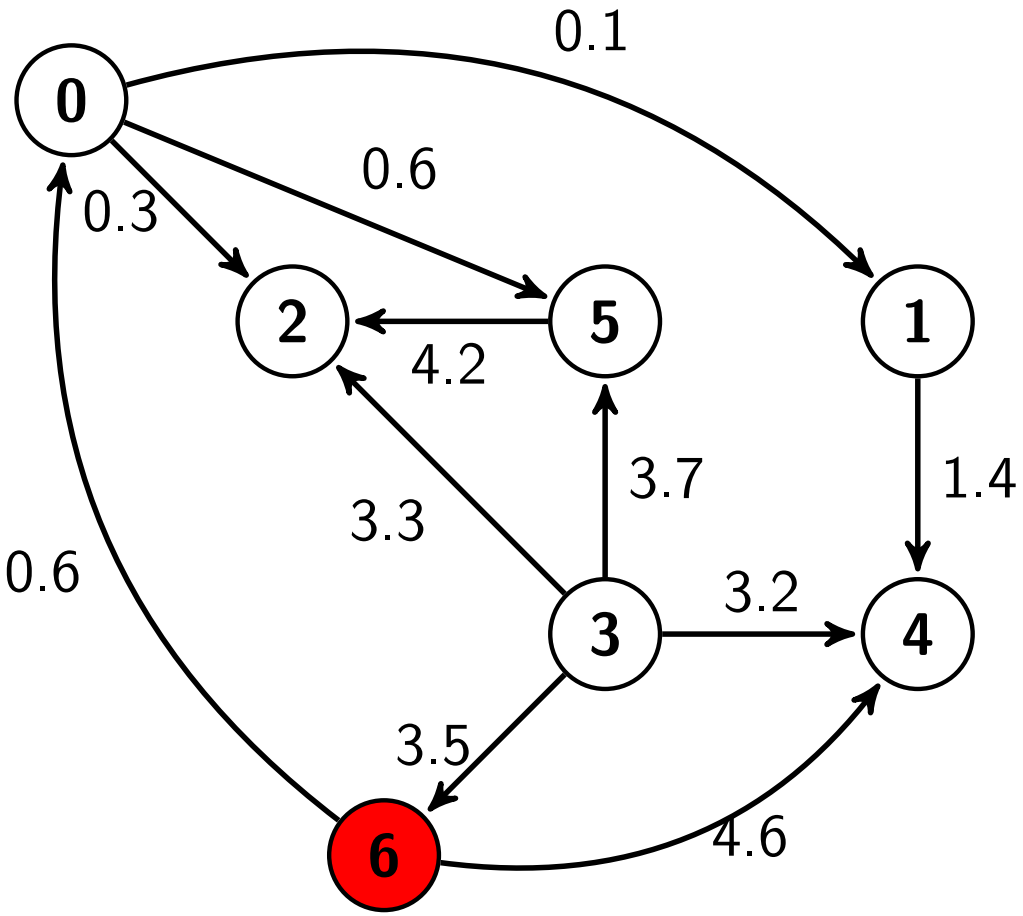
- {**4, 1, 2, 5, 0, 6, 3**}

	distPara	**arestaPara**
0	inf	
1	inf	
2	inf	
3	inf	
4	inf	
5	inf	
6	inf	

- **Origem** é o vértice 6

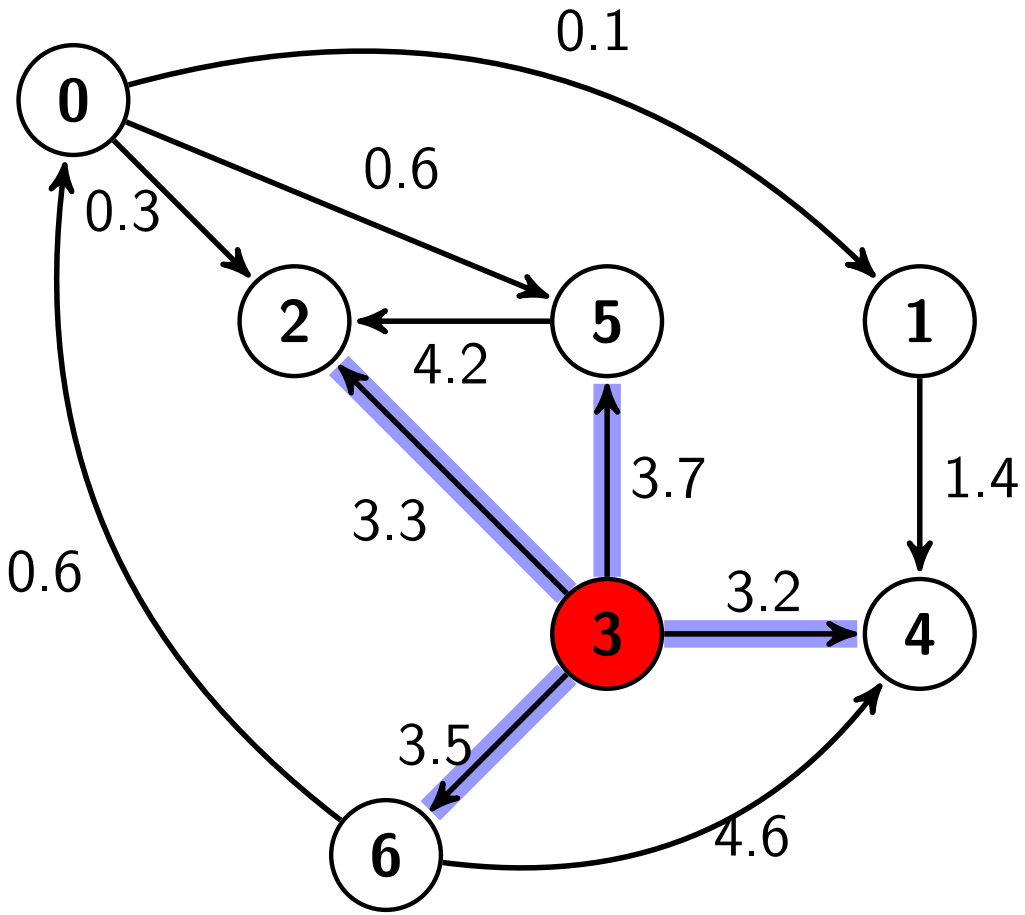
Ordem topológica - pilha

- $\{4, 1, 2, 5, 0, 6, 3\}$



	distPara	arestaPara
0	inf	
1	inf	
2	inf	
3	inf	
4	inf	
5	inf	
6	0	

- Origem é o vértice 6
- **Distância à origem é 0**

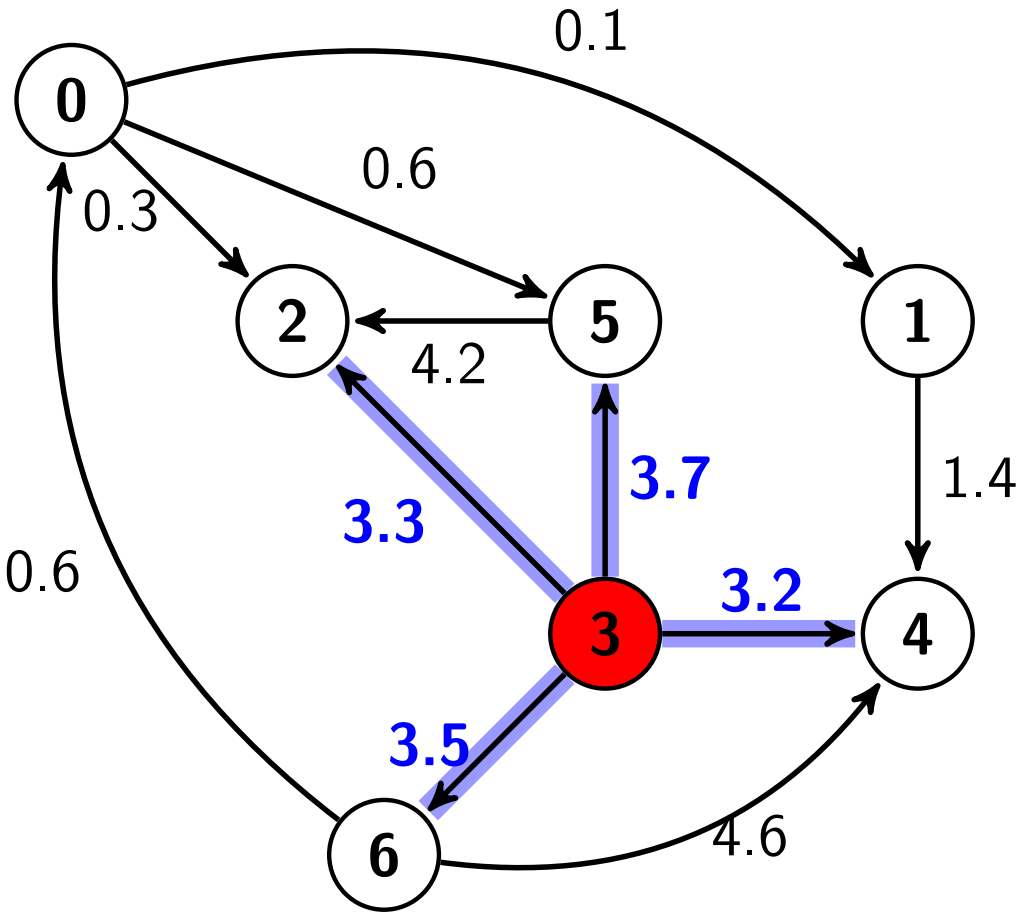


Ordem topológica - pilha

- {**4, 1, 2, 5, 0, 6, 3**}

	distPara	arestaPara
0	inf	
1	inf	
2	inf	
3	inf	
4	inf	
5	inf	
6	0	

- Pegar da pilha um vértice: **3**
- Expandir arestas de **3** para **2**, **4**, **5**, **6**



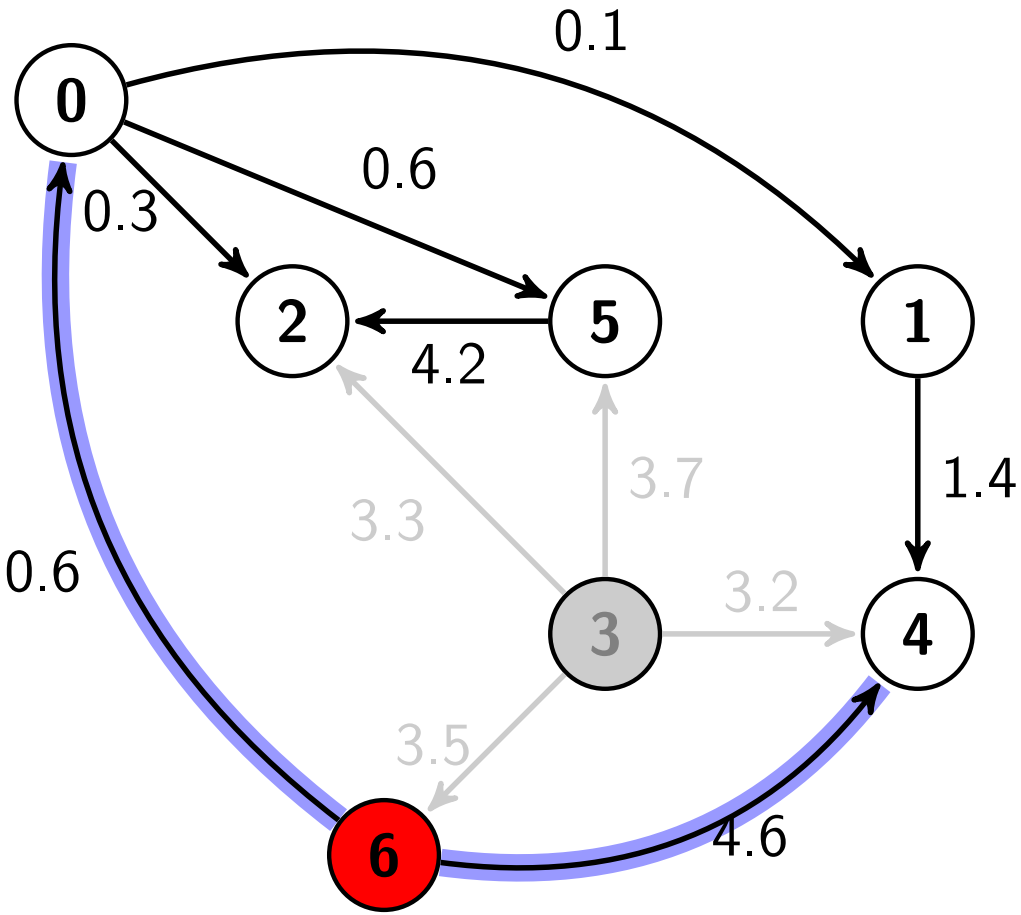

```
1 if (distPara[w]>distPara[v]+a.peso()){
2   distPara[w] = distPara[v] + a.peso();
3   arestaPara[w] = a;
4 }
```

Ordem topológica - pilha

- {**4**, **1**, **2**, **5**, **0**, **6**, **3**}

	distPara	arestaPara
0	inf	
1	inf	
w **2**	inf	
v **3**	inf	
w **4**	inf	
w **5**	inf	
w **6**	0	

- Expandir arestas de **3** para **2**, **4**, **5**, **6**
- Tabela **não** é alterada.

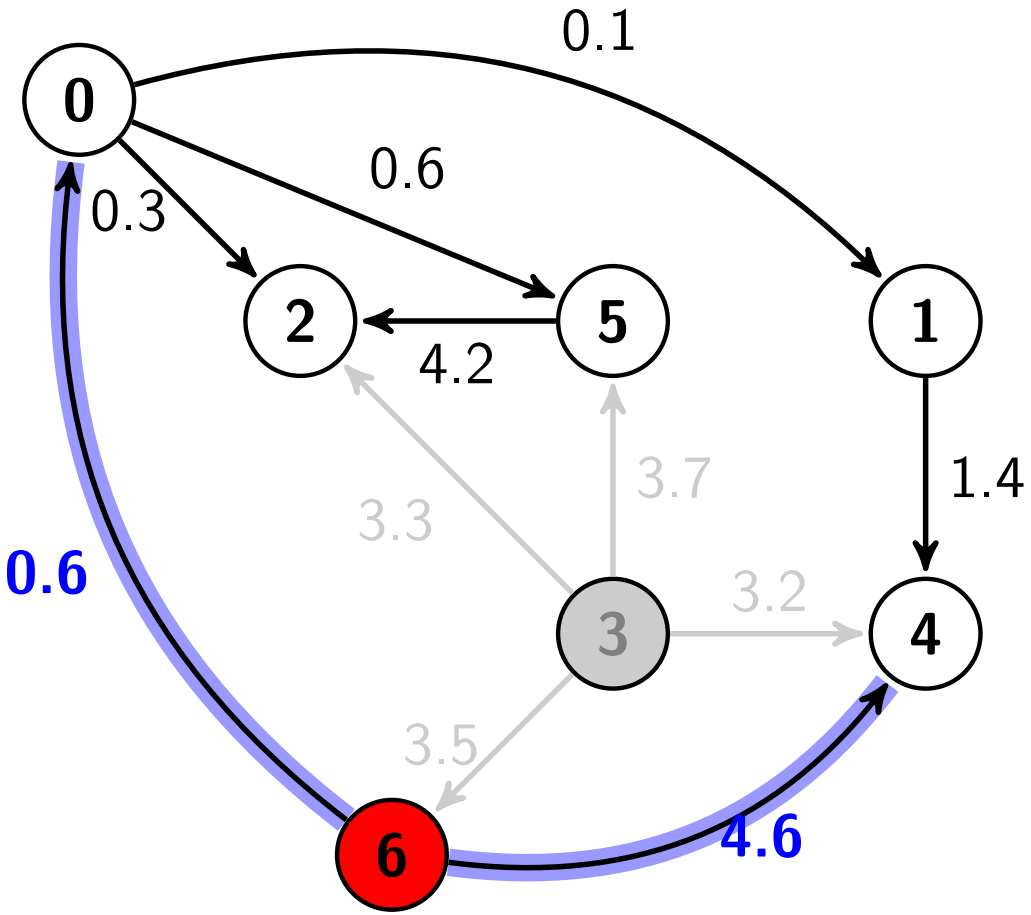


Ordem topológica - pilha

- {**4, 1, 2, 5, 0, 6**}

	distPara	arestaPara
0	inf	
1	inf	
2	inf	
3	inf	
4	inf	
5	inf	
6	0	

- Pegar da pilha um vértice: **6**
- Expandir arestas de **6** para **0** e **4**



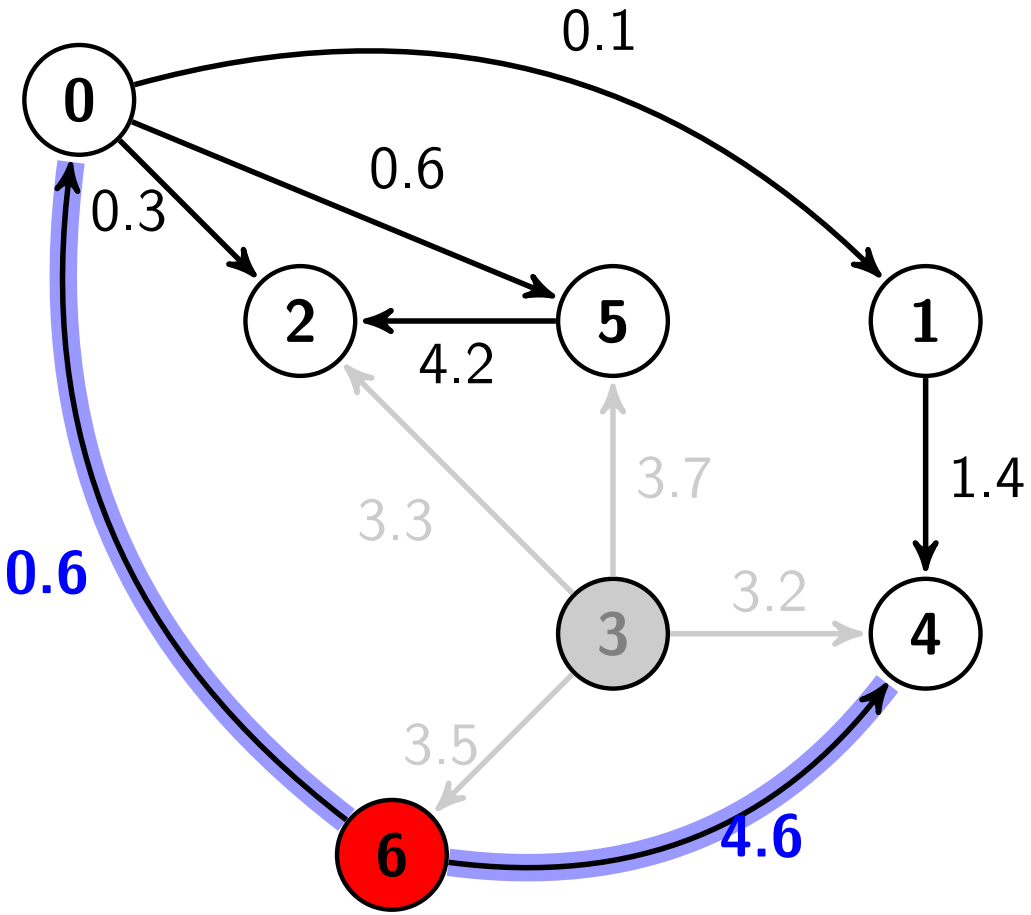
Ordem topológica - pilha

- {**4, 1, 2, 5, 0, 6**}

	distPara	arestaPara
w **0**	inf	
1	inf	
2	inf	
3	inf	
w **4**	inf	
5	inf	
v **6**	0	

```
1 if (distPara[w]>distPara[v]+a.peso()){
2   distPara[w] = distPara[v] + a.peso();
3   arestaPara[w] = a;
4 }
```

- Expandir de **6** para **0** e **4**



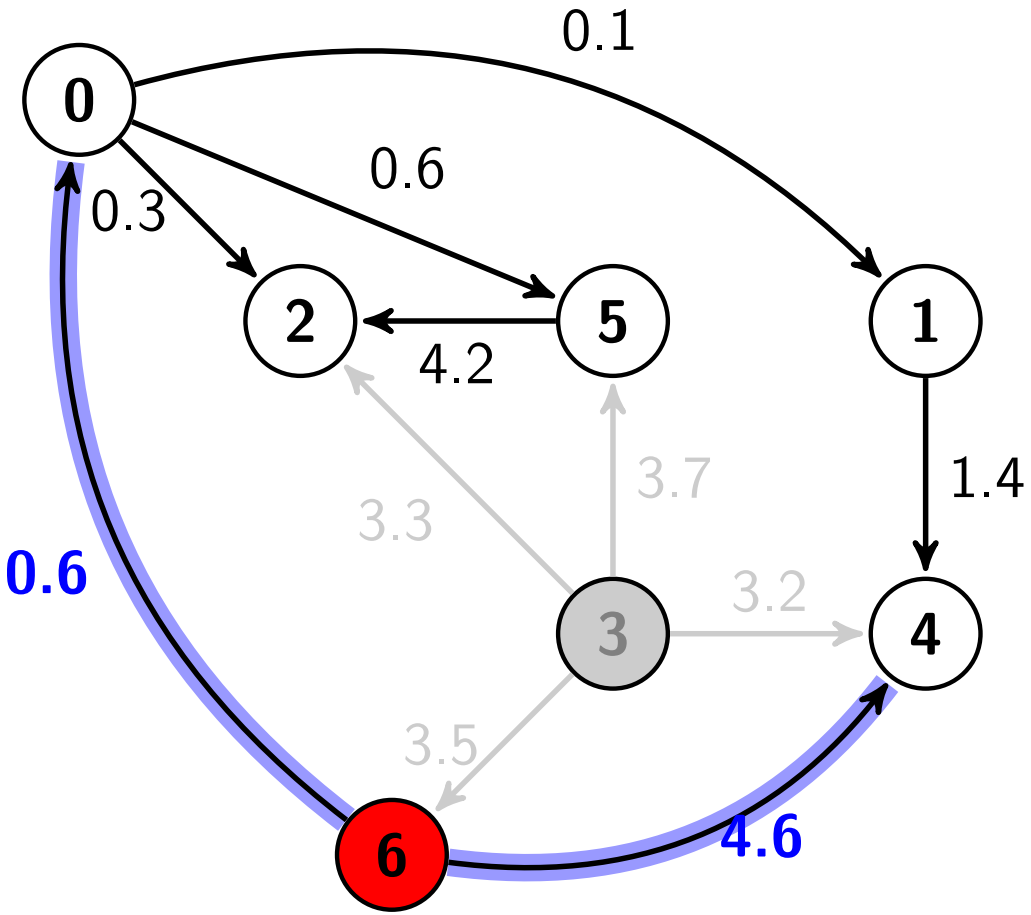
Ordem topológica - pilha

- {**4, 1, 2, 5, 0, 6**}

	distPara	arestaPara
w **0**	**0.6**	**6**
1	inf	
2	inf	
3	inf	
w **4**	inf	
5	inf	
v **6**	0	

```
1 if (distPara[w]>distPara[v]+a.peso()){
2   distPara[w] = distPara[v] + a.peso();
3   arestaPara[w] = a;
4 }
```

- Expandir de **6** para **0** e **4**



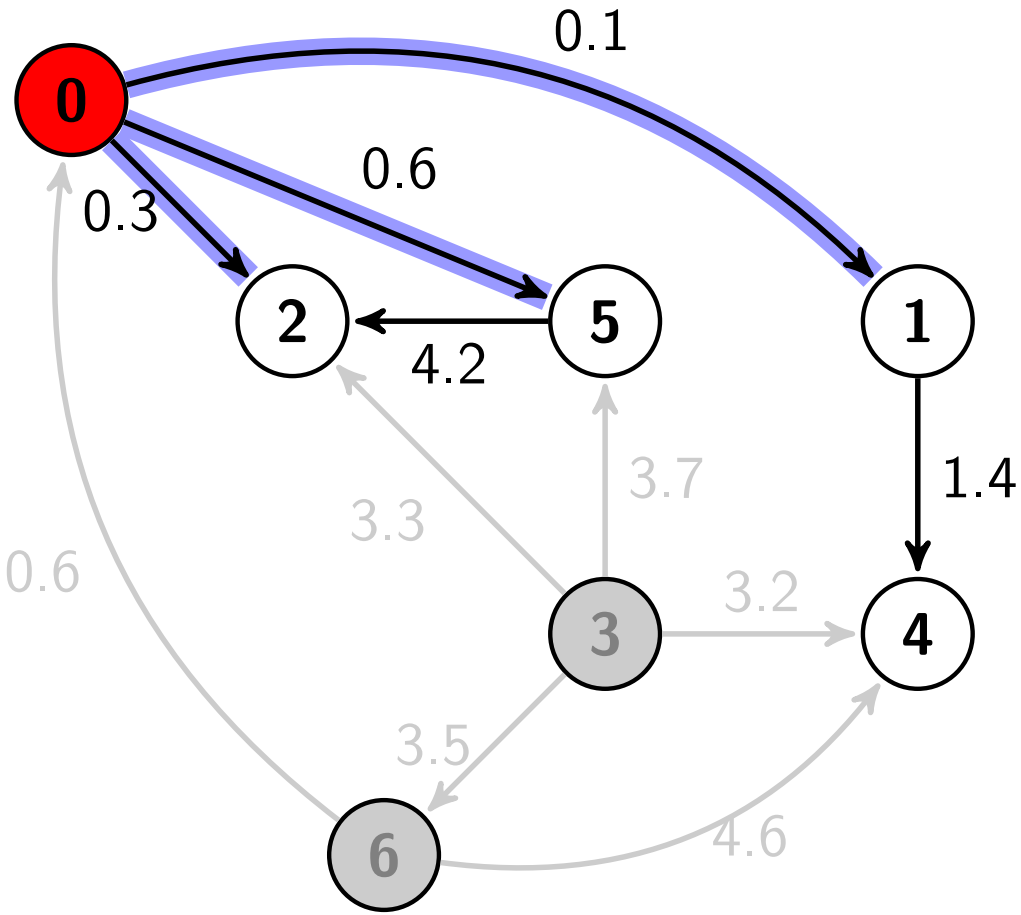
Ordem topológica - pilha

- {**4, 1, 2, 5, 0, 6**}

	distPara	arestaPara
w **0**	**0.6**	**6**
1	inf	
2	inf	
3	inf	
w **4**	**4.6**	**6**
5	inf	
v **6**	0	

```
1 if (distPara[w]>distPara[v]+a.peso()){
2   distPara[w] = distPara[v] + a.peso();
3   arestaPara[w] = a;
4 }
```

- Expandir de **6** para **0** e **4**

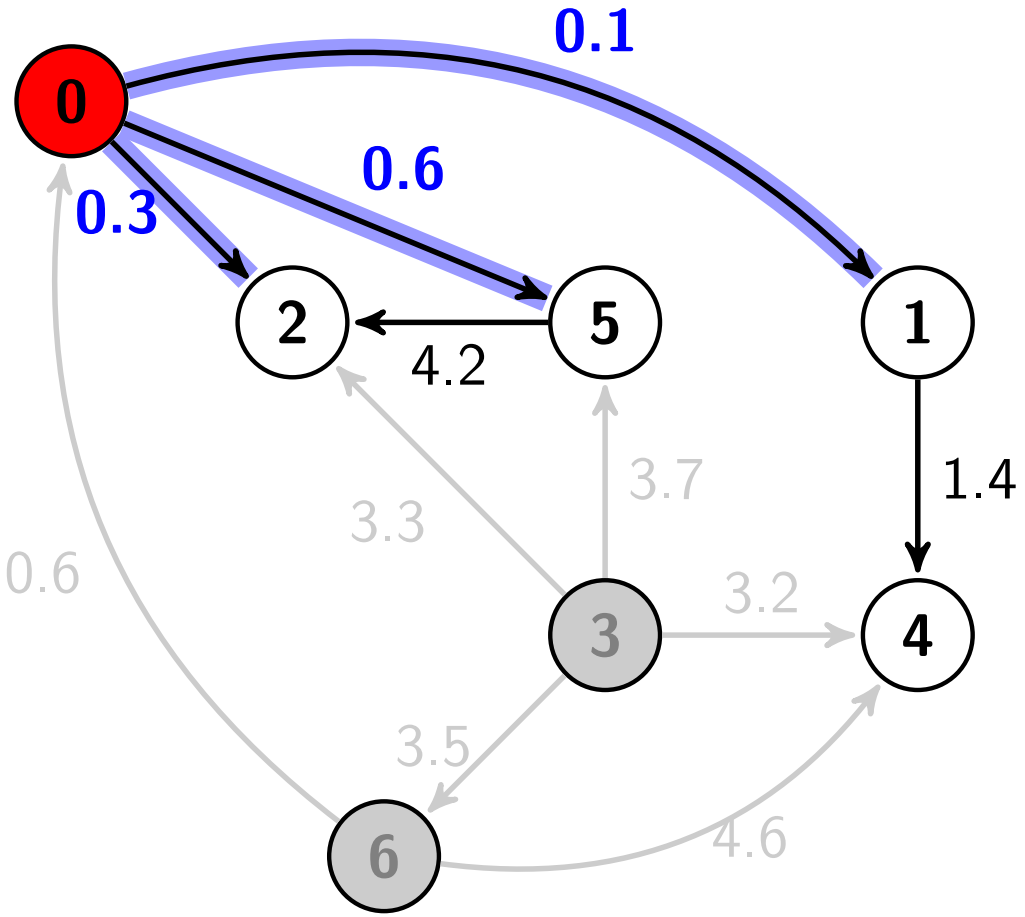


Ordem topológica - pilha

- {**4, 1, 2, 5, 0**}

	distPara	arestaPara
0	0.6	6
1	inf	
2	inf	
3	inf	
4	4.6	6
5	inf	
6	0	

- Pegar da pilha um vértice: **0**
- Expandir arestas de **0** para **1**, **2** e **5**



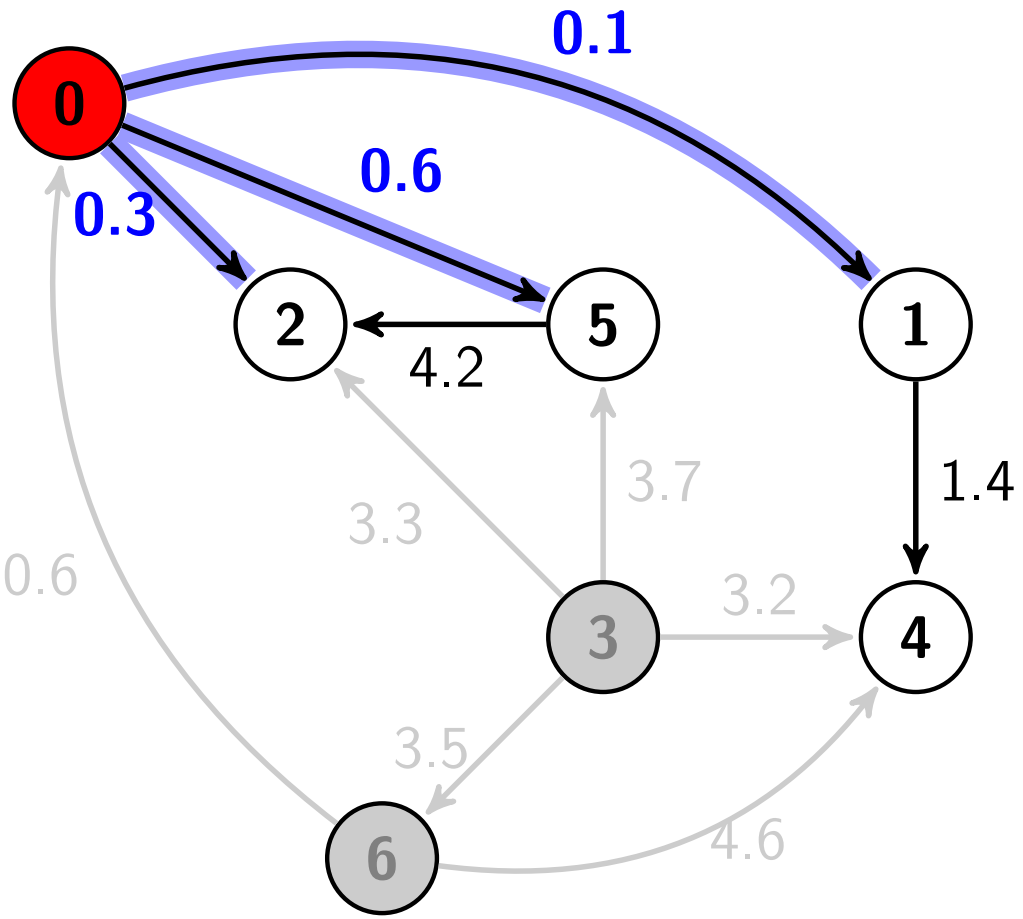
Ordem topológica - pilha

- {4, 1, 2, 5, 0}

	distPara	arestaPara
v 0	0.6	6
w 1	0.6 + 0.1	0
w 2	0.6 + 0.3	0
3	inf	
4	4.6	6
w 5	0.6 + 0.6	0
6	0	

```
1 if (distPara[w]>distPara[v]+a.peso()){
2   distPara[w] = distPara[v] + a.peso();
3   arestaPara[w] = a;
4 }
```

- Expandir de 0 para 1, 2 e 5



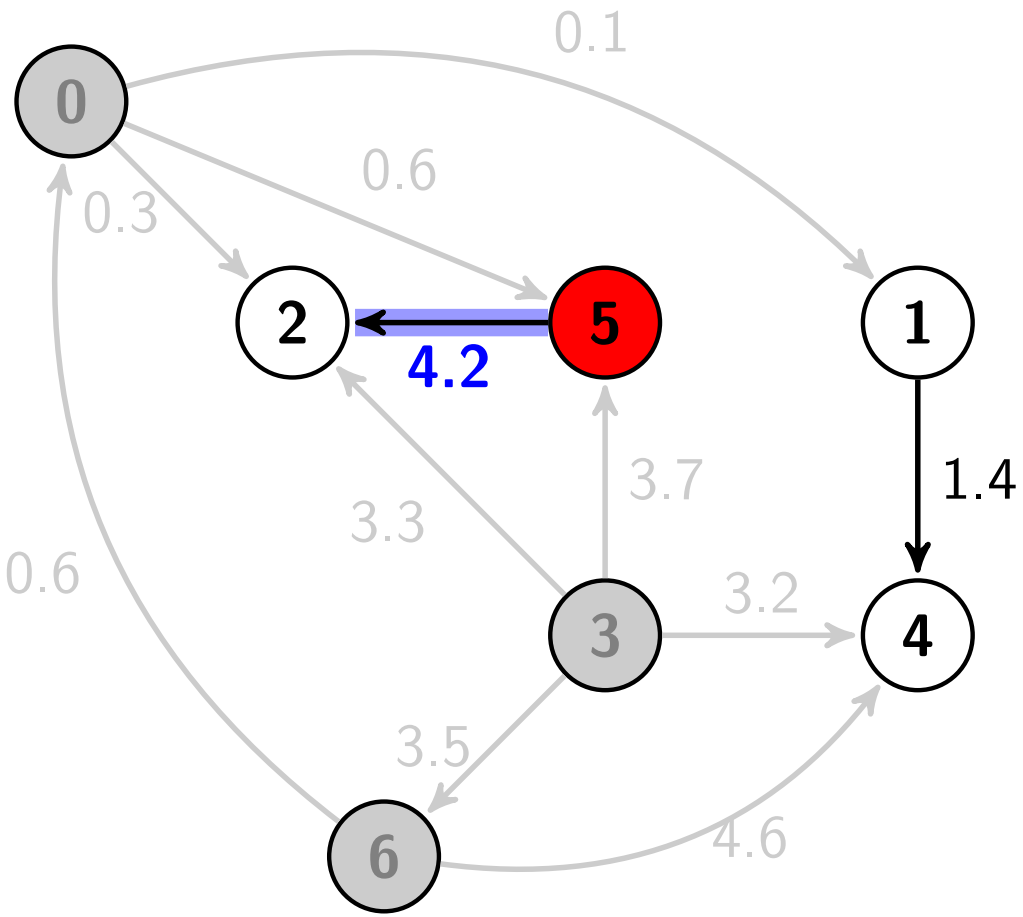
Ordem topológica - pilha

- {**4**, **1**, **2**, **5**, **0**}

	distPara	arestaPara
0	0.6	6
1	0.7	0
2	0.9	0
3	inf	
4	4.6	6
5	1.2	0
6	0	

```
1 if (distPara[w]>distPara[v]+a.peso()){
2   distPara[w] = distPara[v] + a.peso();
3   arestaPara[w] = a;
4 }
```

- Expandir de **0** para **1**, **2** e **5**

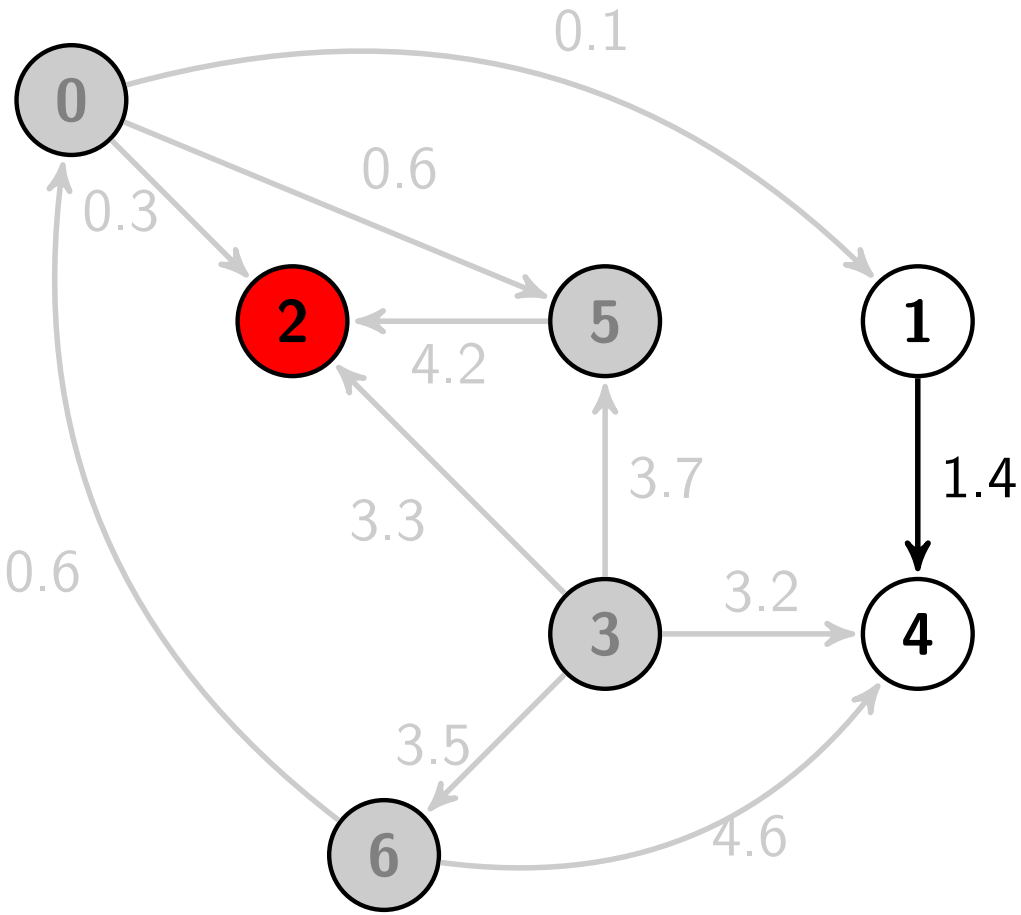


Ordem topológica - pilha

- {**4**, **1**, **2**, **5**}

	distPara	arestaPara
0	0.6	6
1	0.7	0
2	0.9	0
3	inf	
4	4.6	6
5	1.2	0
6	0	

- Pegar da pilha um vértice: **5**
- Expandir arestas de **5** para **2**.
- Não muda nada: `0.9 < 1.2+4.2`!

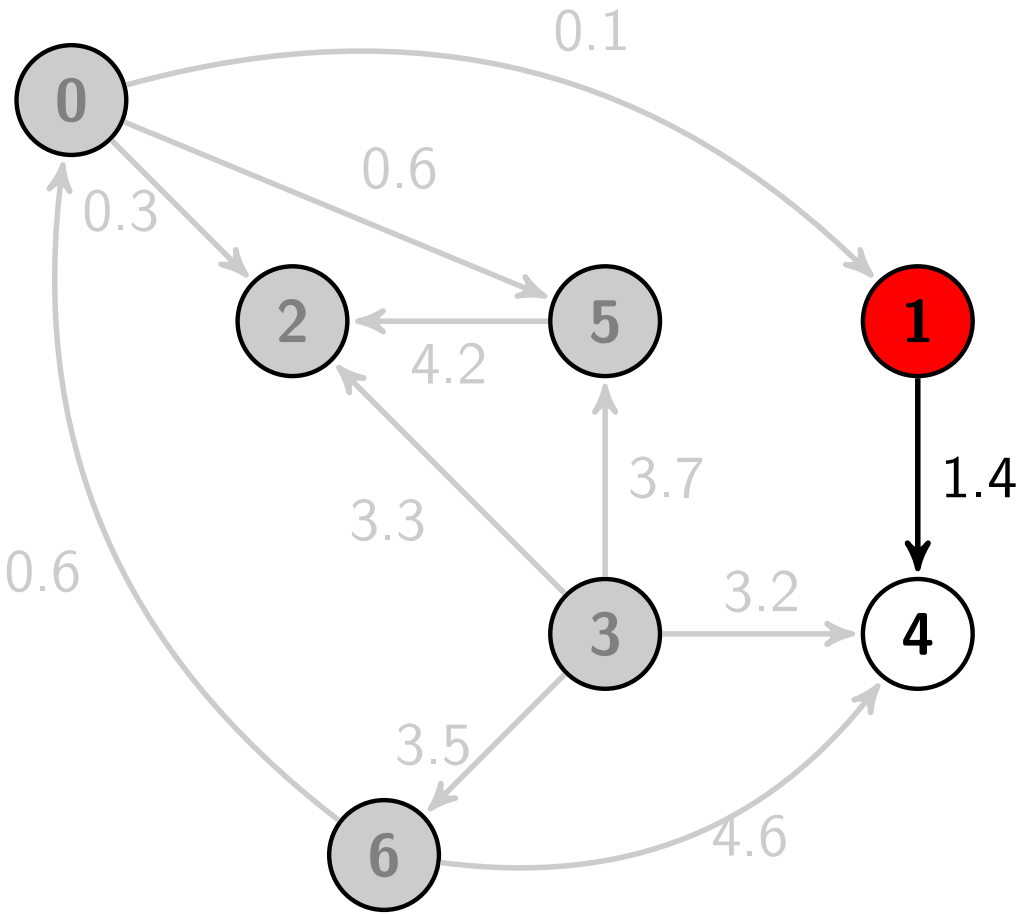


Ordem topológica - pilha

- $\{4, 1, 2\}$

	distPara	arestaPara
0	0.6	6
1	0.7	0
2	0.9	0
3	inf	
4	4.6	6
5	1.2	0
6	0	

- Pegar da pilha um vértice: **2**
- **Não** tem para onde expandir.

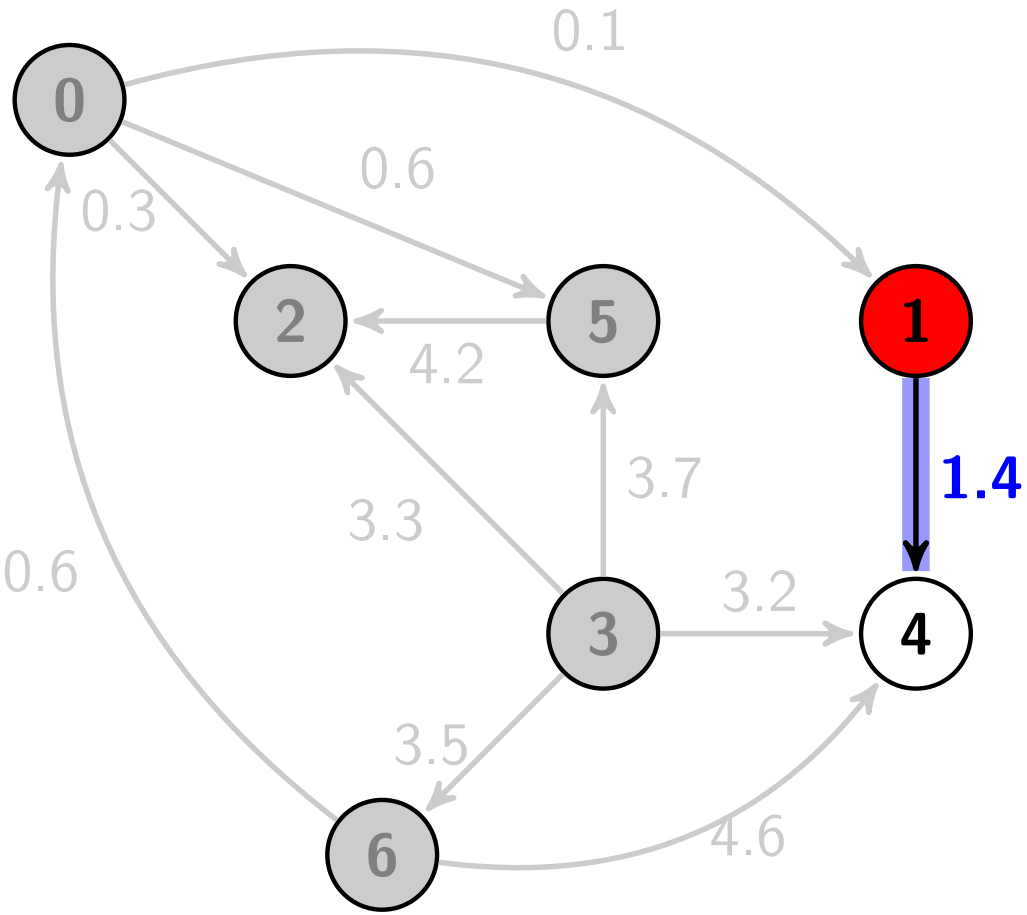


Ordem topológica - pilha

- {**4**, **1**}

	distPara	arestaPara
0	0.6	6
1	0.7	0
2	0.9	0
3	inf	
4	4.6	6
5	1.2	0
6	0	

- Pegar da pilha um vértice: **1**
- Expandir de **1** para **4**



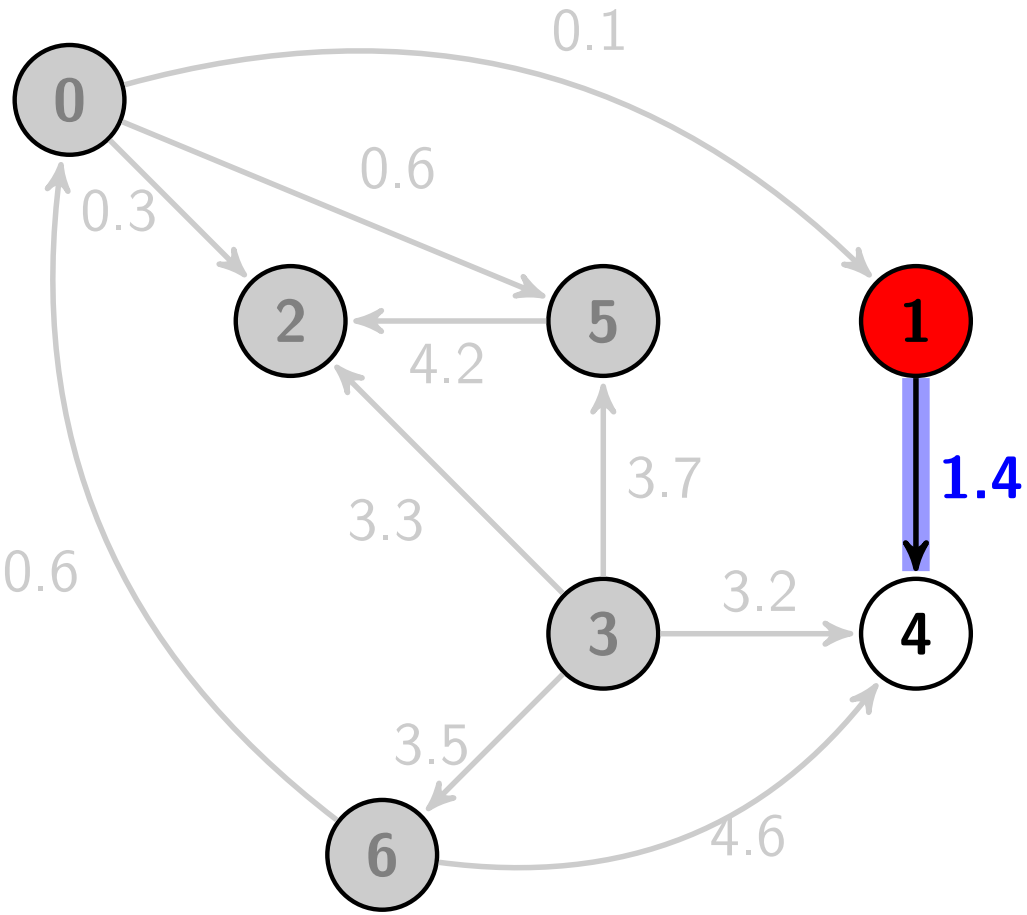
Ordem topológica - pilha

- {**4**, **1**}

	distPara	arestaPara
0	0.6	6
v 1	0.7	0
2	0.9	0
3	inf	
w 4	4.6	6
5	1.2	0
6	0	

```
1 if (distPara[w]>distPara[v]+a.peso()){
2   distPara[w] = distPara[v] + a.peso();
3   arestaPara[w] = a;
4 }
```

- Expandir de **1** para **4**.
- `4.6 > 0.7 + 1.4`, atualizar tabela.



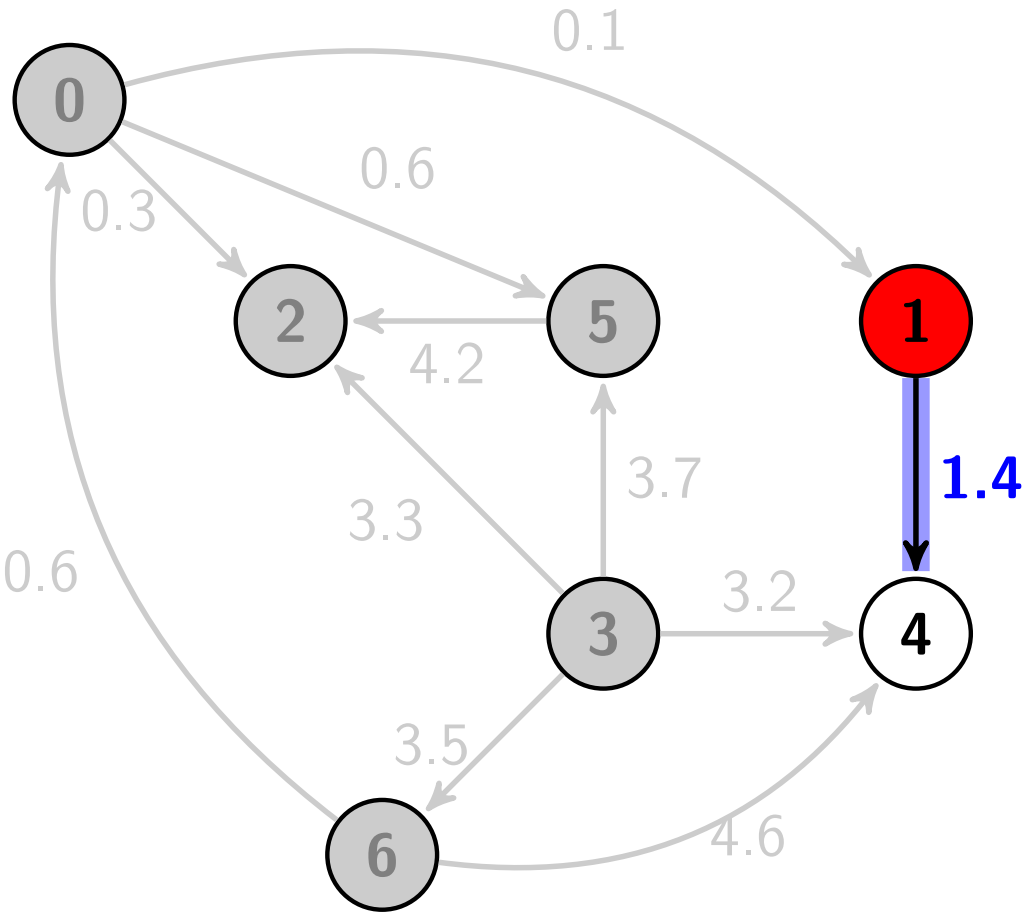
Ordem topológica - pilha

- $\{4, 1\}$

	distPara	arestaPara
0	0.6	6
v 1	0.7	0
2	0.9	0
3	inf	
w 4	0.7 + 1.4	1
5	1.2	0
6	0	

```
1 if (distPara[w]>distPara[v]+a.peso()){
2   distPara[w] = distPara[v] + a.peso();
3   arestaPara[w] = a;
4 }
```

- Expandir de **1** para **4**.
- `4.6 > 0.7 + 1.4`, atualizar tabela.



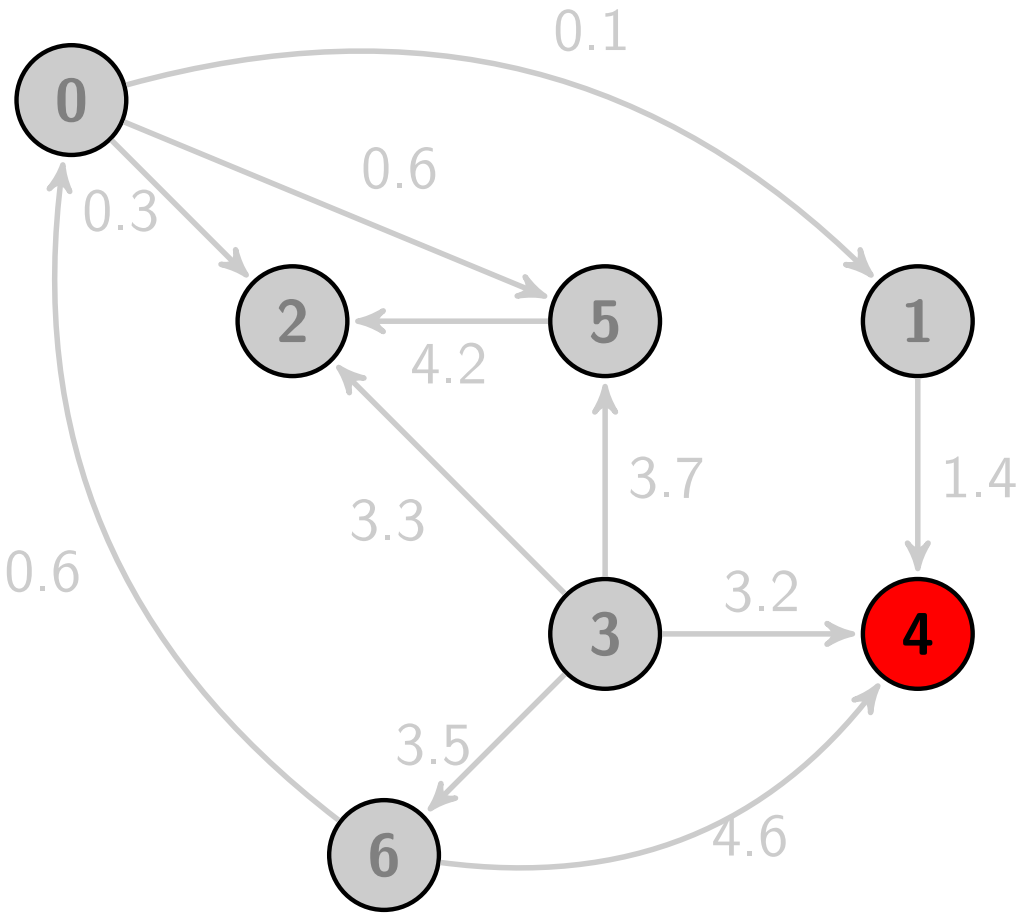
Ordem topológica - pilha

- {**4**, **1**}

	distPara	arestaPara
0	0.6	6
v 1	0.7	0
2	0.9	0
3	inf	
w 4	2.1	**1**
5	1.2	0
6	0	

```
1 if (distPara[w]>distPara[v]+a.peso()){
2   distPara[w] = distPara[v] + a.peso();
3   arestaPara[w] = a;
4 }
```

- Expandir de **1** para **4**.
- `4.6 > 0.7 + 1.4`, atualizar tabela.

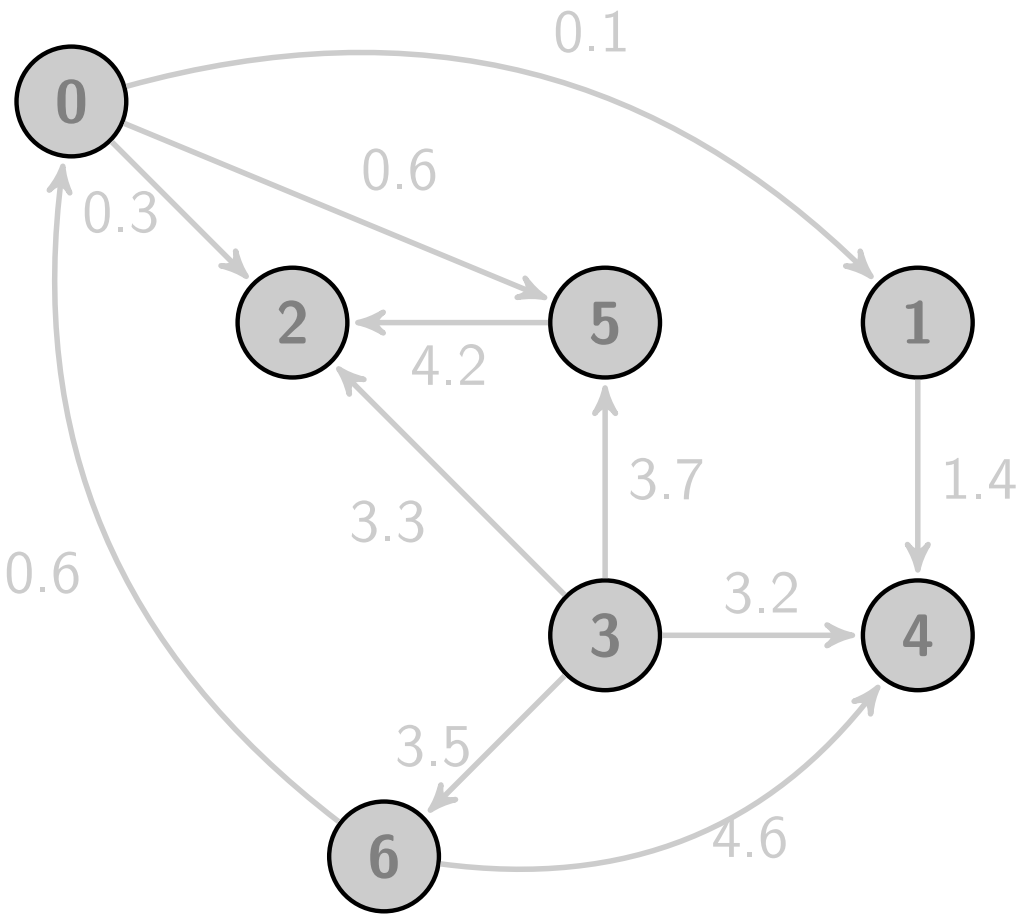


Ordem topológica - pilha

- {**4**}

	distPara	arestaPara
0	0.6	6
1	0.7	0
2	0.9	0
3	inf	
v 4	2.1	1
5	1.2	0
6	0	

- Pegar da pilha um vértice: **4**
- **Não** tem para onde expandir



Ordem topológica - pilha

- {}

	distPara	arestaPara
0	0.6	6
1	0.7	0
2	0.9	0
3	inf	
4	2.1	1
5	1.2	0
6	0	

- Pilha está vazia. Fim do algoritmo.
- Menores caminhos a partir de **6** estão prontos para serem consultados.

Aplicações - Ordenação topológica

- Detecção de ciclos em grafos
- Verificar se sequência de atividades é possível de respeitar
- Paralelização de algoritmos
- Linguagem de programação: herança cíclica em Java
- Recálculo de planilhas de dados

Aplicações - Menor caminho

- Roteamento em mapas (GPS)
- Gerência de projetos - caminho crítico
- Redimensionamento de imagens
- Planejamento de tráfego urbano
- Protocolos de rede (OSPF, BGP)
- Comércio de moedas estrangeiras