Grafos

Representações, Buscas e Aplicações

MC 202 – Estruturas de Dados

Prof. Anderson Rocha

Organização dessa aula

Sumário

- Recapitulando
- Implementação de Grafos
- Algoritmos em Grafos: Busca em Profundidade e Largura
- Aplicações
- Lição da aula de hoje
- Próximos capítulos
- Exercícios

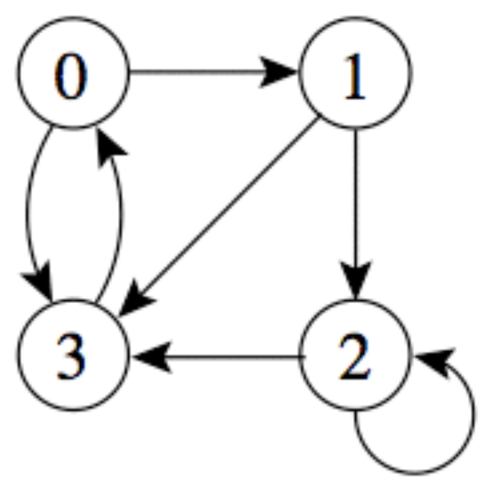
Warming Up...

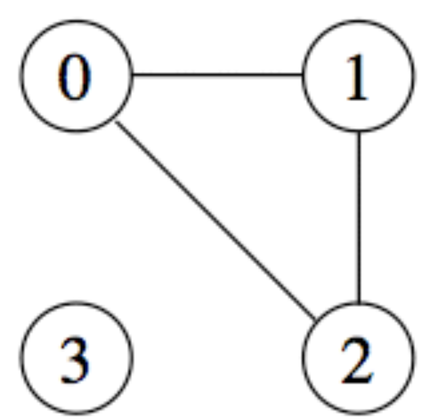
O que vimos até agora?

- O que vimos no curso até o momento
  - Arranjos, filas, pilhas
  - Árvores (Binária, B, heaps)
  - Hashes
  - Grafos (Definições)

Grafos – Definições

- Um grafo G = (V,A) é constituído de um conjunto de vértices e um conjunto de arestas conectando pares de vértices.
- Pode ser direcionado, não direcionado, ponderado





Grafos – Definições

- Muitas aplicações em computação precisam considerar um conjunto de conexões entre pares de objetos
  - Existe um caminho para ir de um objeto a outro seguindo as conexões?
  - Qual a menor distância entre um objeto e outro?
  - Quantos objetos podem ser alcançados a partir de um determinado objeto?

Grafos – Aplicações

- Alguns problemas práticos que podemos resolver com grafos
  - Ajudar máquinas de busca a localizar informação relevante na Web
  - Descobrir os melhores casamentos entre posições disponíveis em empresas e pessoas que aplicaram para as posições de interesse
  - Descobrir qual o roteiro mais curto para visitar as principais cidades de uma regiões turística

Grafos – Modelagem

- Finalmente, vimos a questão da modelagem de problemas segundo a ótica de grafos
- Exemplo da conexão entre cidades, distâncias, pedágios, pavimentação etc.

Ok... mas como implementar?

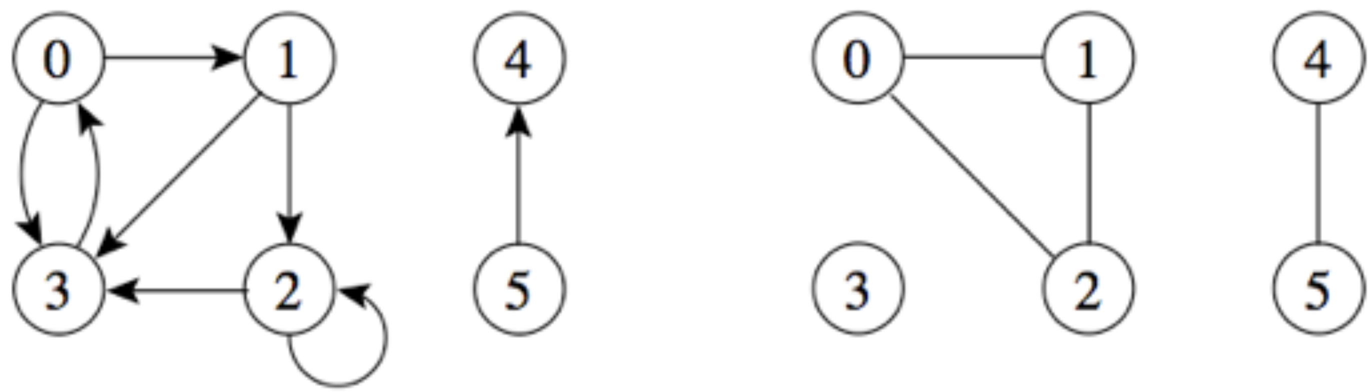
Idéias básicas

- Podemos implementar o tipo abstrato de dados "Grafo" basicamente de duas formas
  - Matriz de Adjacências
  - Lista de Adjacências

Matriz de Adjacências

- A matriz de adjacências de um grafo $G = (V, A)$ contendo n vértices é uma matriz de $n \times n$
- $A[i,j]$ é 1 se e somente se existe um arco do vértice i para o vértice j.

Matriz de Adjacências



	0	1	2	3	4	5
0		1		1		
1			1	1		
2			1	1		
3	1					
4						
5						

(a)

	0	1	2	3	4	5
0		1	1			
1	1		1			
2	1	1				
3						
4						
5						

(b)

Como fazer com grafos ponderados?

Matriz de Adjacências

- Quando devemos optar por essa representação?
  - Grafos densos ($|A| \sim |V|^2$)
  - Por que?
- O tempo necessário para acessar um elemento é independente de $|V|$ ou $|A|$

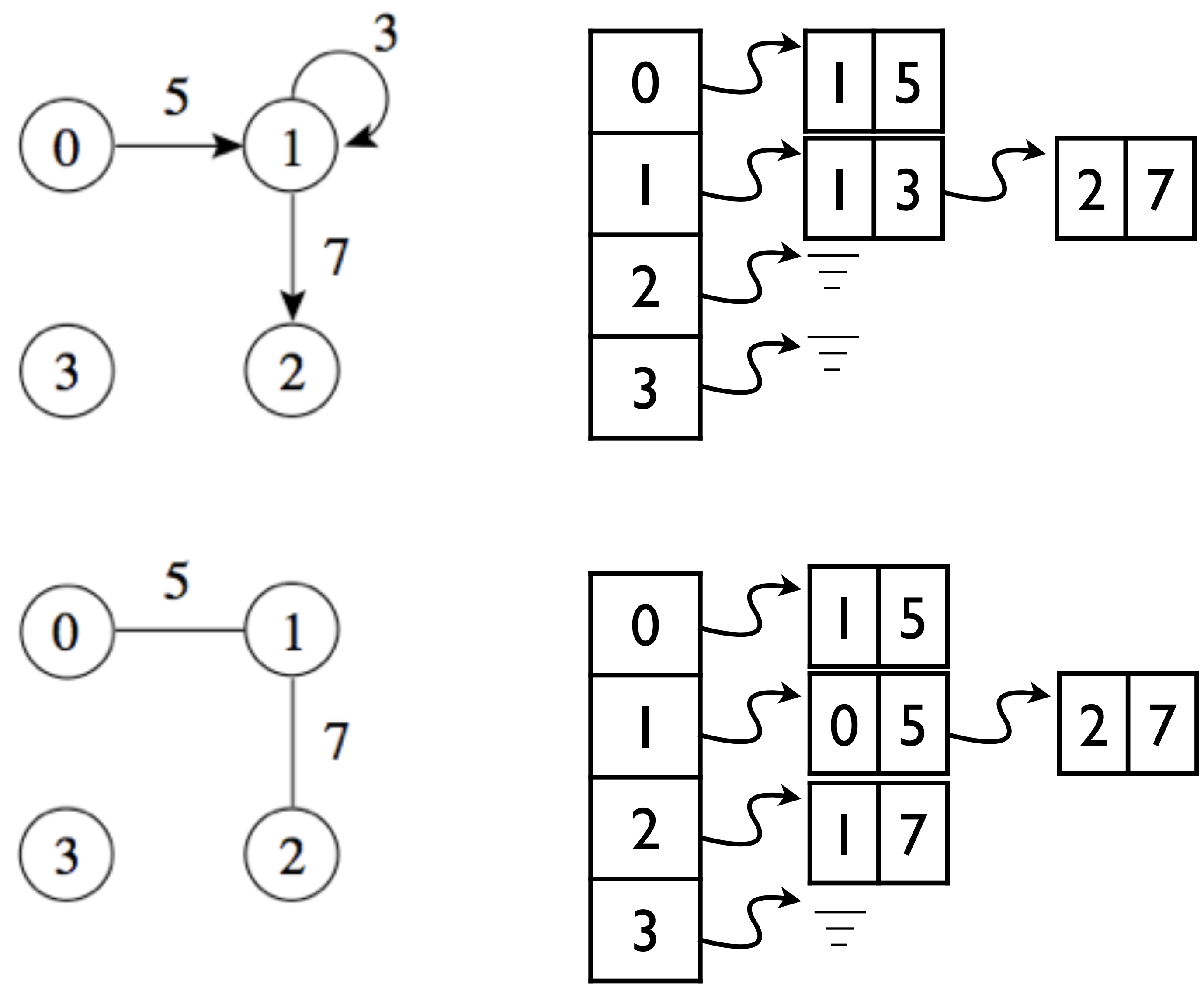
Matriz de Adjacências

- É útil quando queremos saber com rapidez se existe um aresta ligando dois vértices
- Qual a desvantagem?
  - Armazenamento necessita $\Omega(|V|^2)$ de espaço
  - Então ler ou examinar a matriz nos custa $O(|V|^2)$

Lista de Adjacências

- Utilizamos listas encadeadas
- Temos um arranjo *adj* de |V| listas, uma para cada vértice em V
- Para cada vértice *u* em V
  - *adj[u]* contém todos os vértices adjacentes a *u* em G.

Lista de Adjacências



Lista de Adjacências

- Armazenamento em ordem arbitrária
- Complexidade de espaço é $O(|V| + |A|)$
- Quando é indicada?
  - Grafos esparsos ($|A| << |V|^2$)

Lista de Adjacências

- Qual a maior desvantagem?
  - Pode ter tempo O(|V|) para determinar se existe uma aresta entre o vértice *i* e o vértice *j*
  - Por que? Podem existir O(|V|) vértices na lista de adjacentes do vértice *i*.

Ok... mas o que podemos fazer com isso?

Algoritmos em Grafos

- Efetuar buscas
- Determinar topologias e restrições
- Achar caminhos mínimos
- Achar componentes

Algoritmos em Grafos

- Determinar comunidades em redes sociais
- Analisar padrões de espalhamento e comportamento
- etc.

Busca em Profundidade

Busca em Profundidade

- É um algoritmo para caminhar no grafo
- A estratégia é buscar o "mais profundo" no grafo sempre que possível
- As arestas são exploradas a partir do vértice *v* mais recentemente descoberto que ainda possui arestas não exploradas saindo dele

Busca em Profundidade

- Quando todas as arestas adjacentes a *v* tiverem sido exploradas a busca "anda para tras" para explorar vértices que saem do vértice do qual *v* foi descoberto
- Algoritmo útil em diversas aplicações
  - Ordenação topológica
  - Componentes conectados
  - Verificação de ciclos

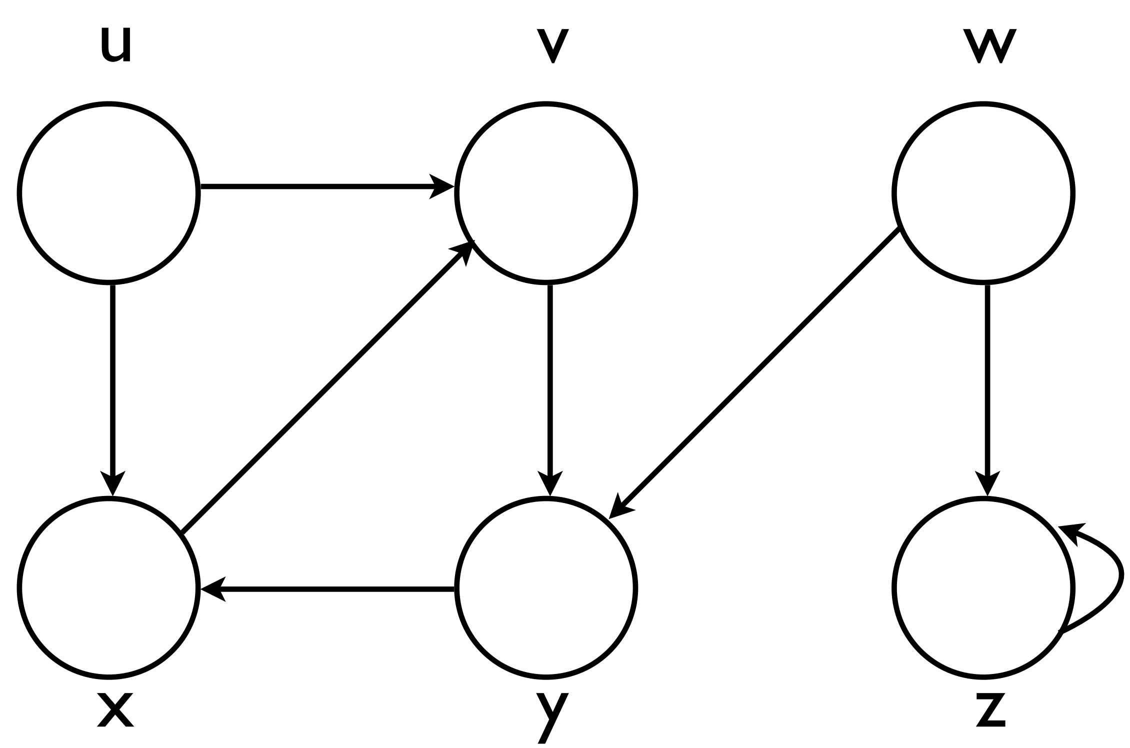
Busca em Profundidade

- Para acompanhar o progresso do algoritmo, cada vértice é colorido de branco, cinza ou preto
- Todos os vértices começam em branco
- Quando um vértice é descoberto pela 1ª vez, ele se torna cinza

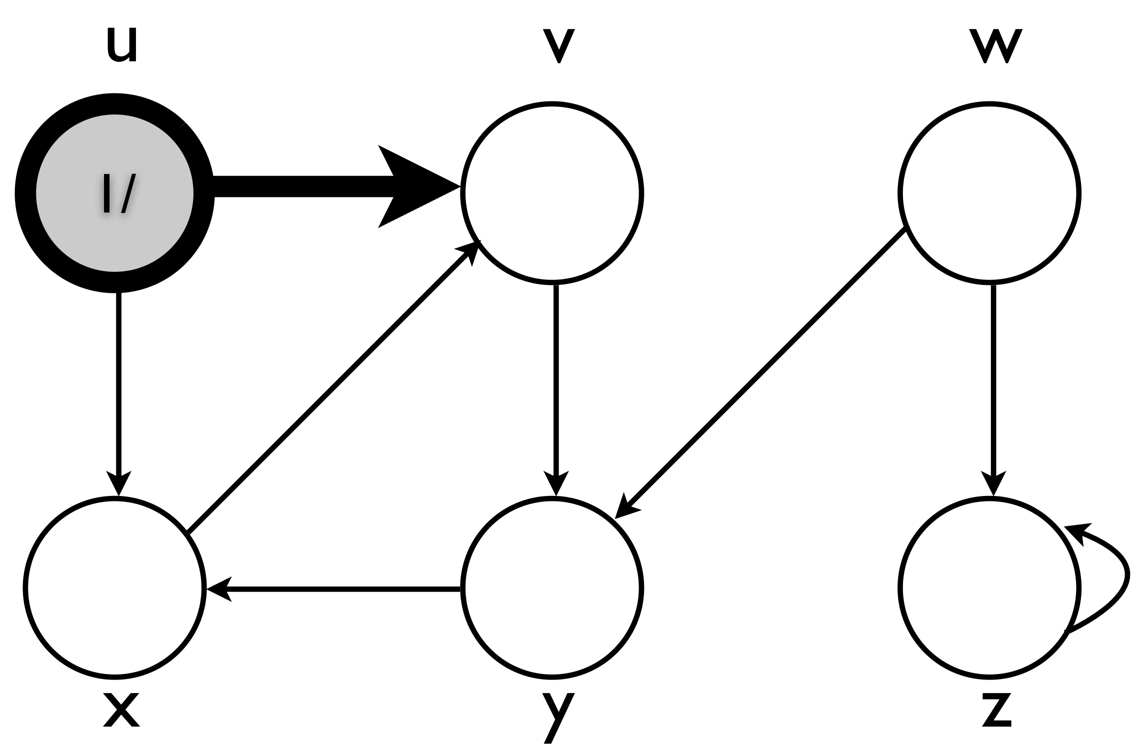
Busca em Profundidade

- Quando acabamos de analisar sua lista de adjacentes ele se torna preto
- Usamos "time-stamps" para marcar o tempo de descoberta e o tempo de finalização

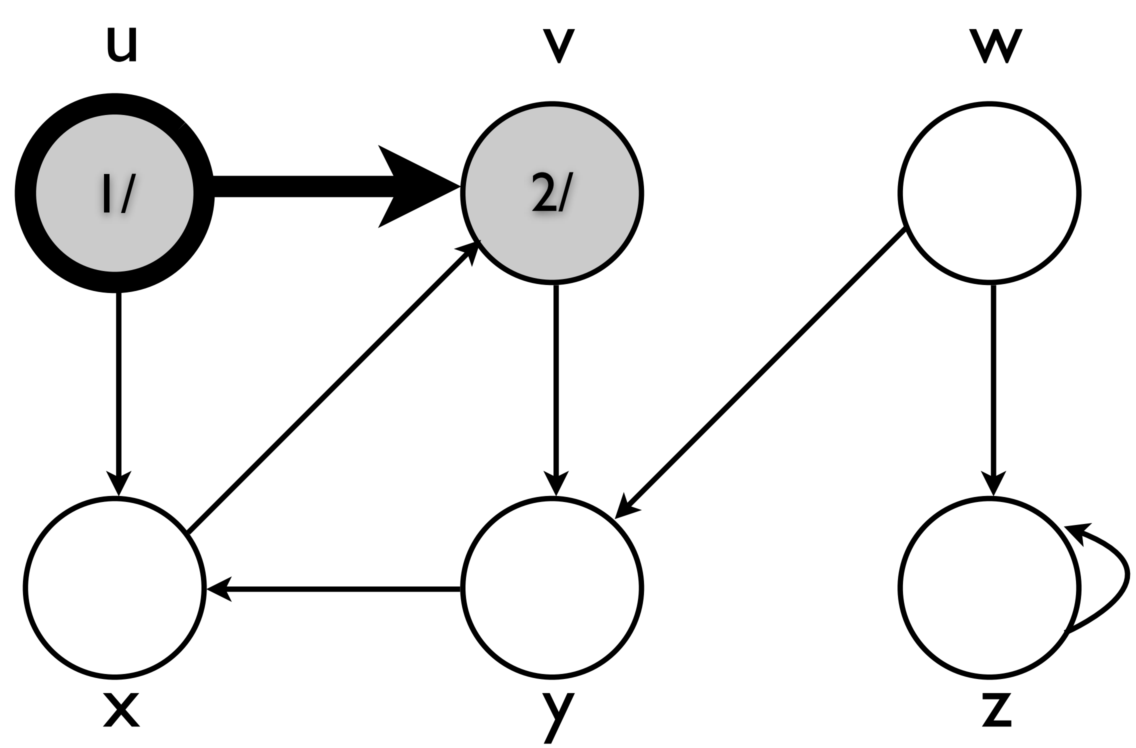
Busca em Profundidade



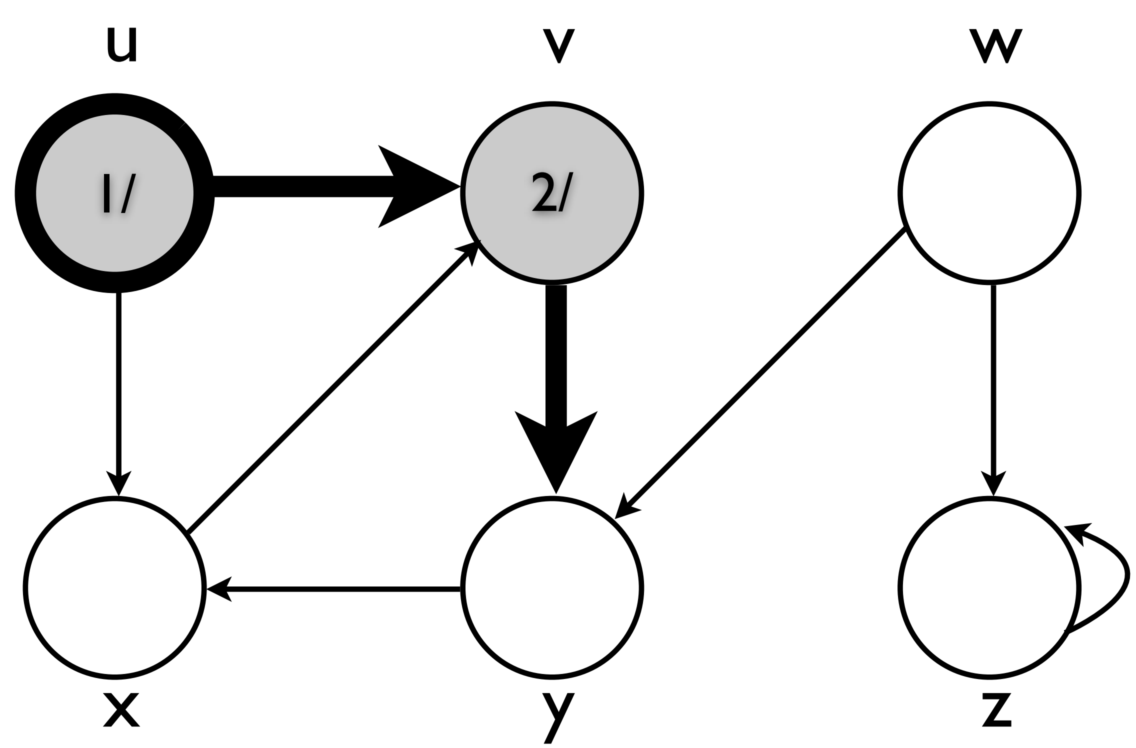
Busca em Profundidade



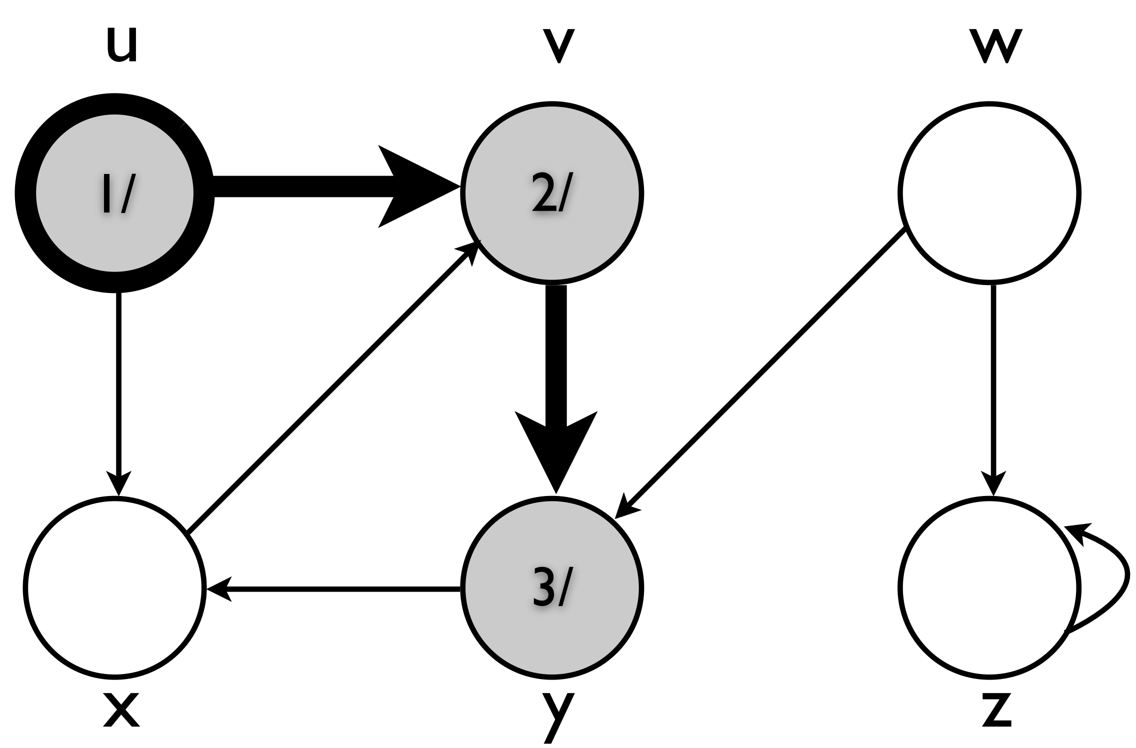
Busca em Profundidade



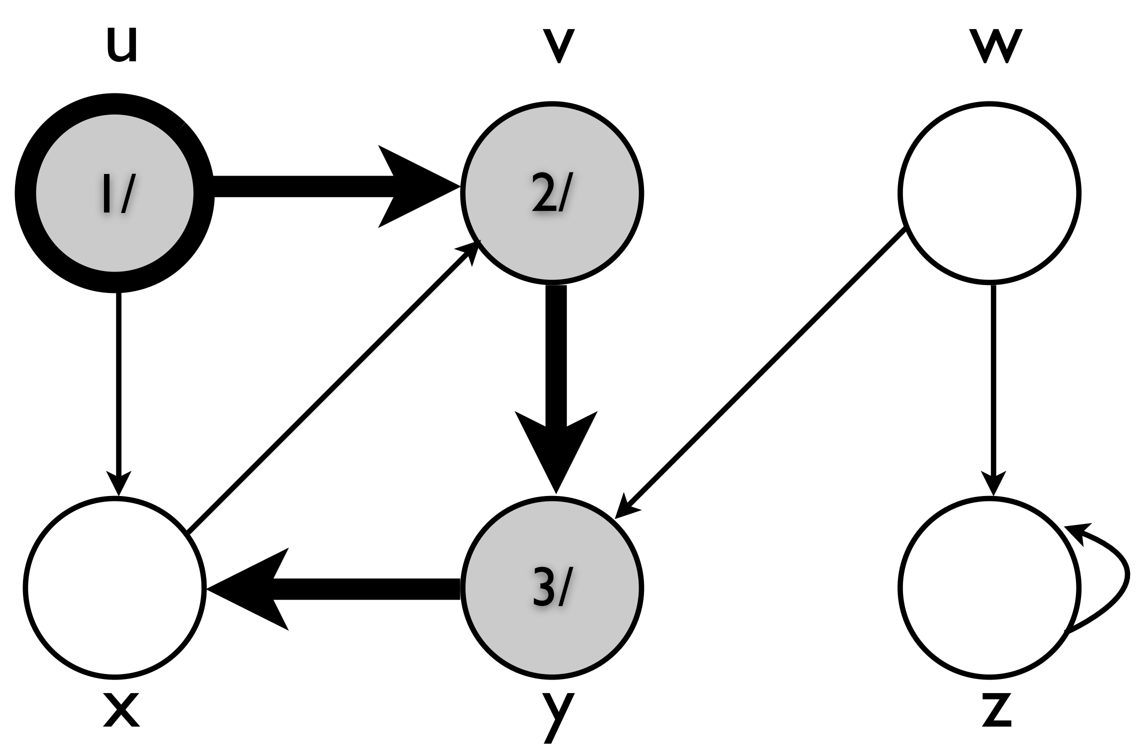
Busca em Profundidade



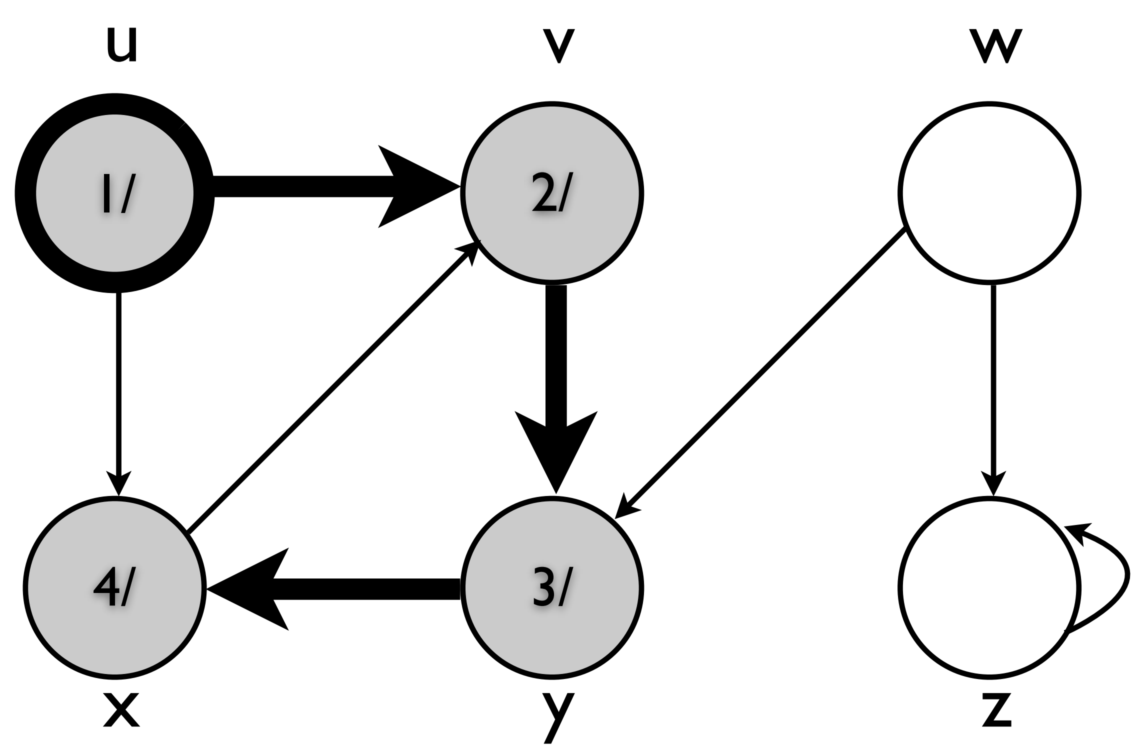
Busca em Profundidade



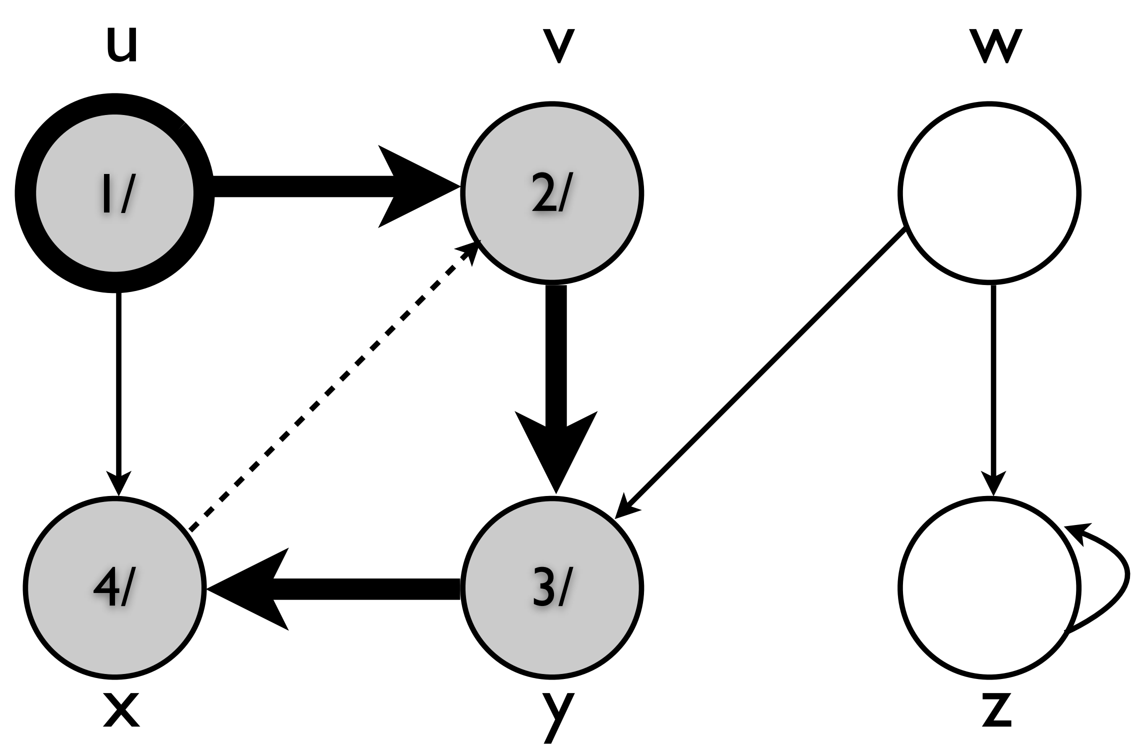
Busca em Profundidade



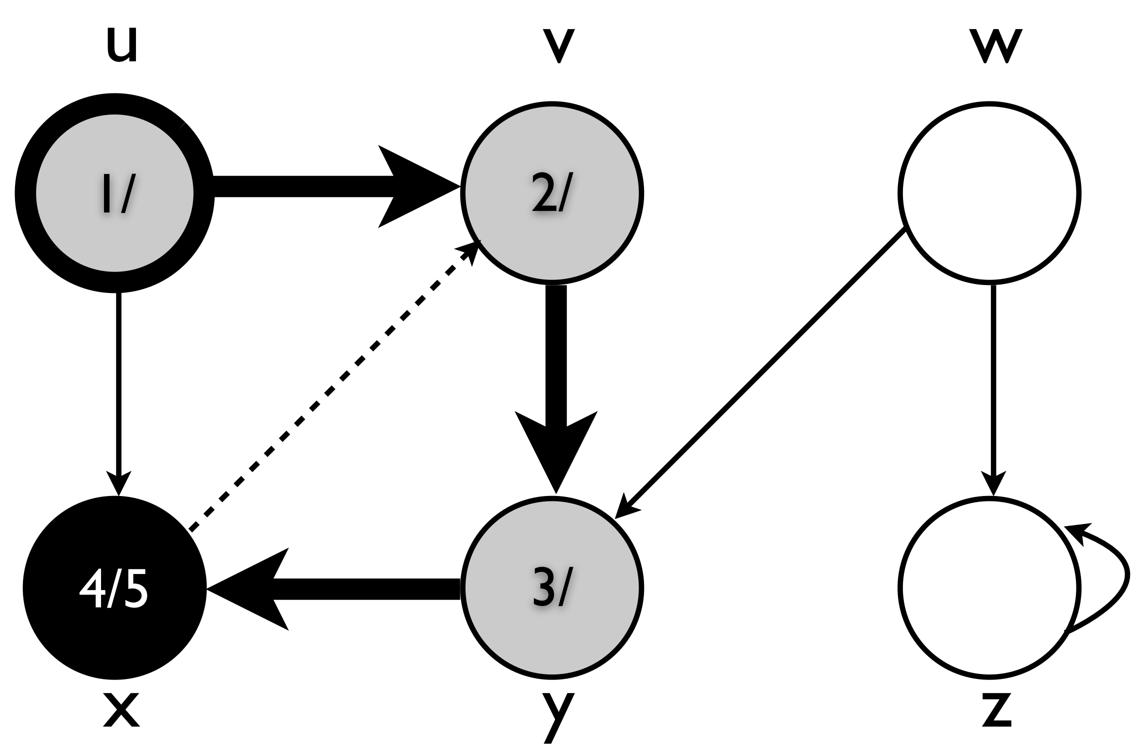
Busca em Profundidade



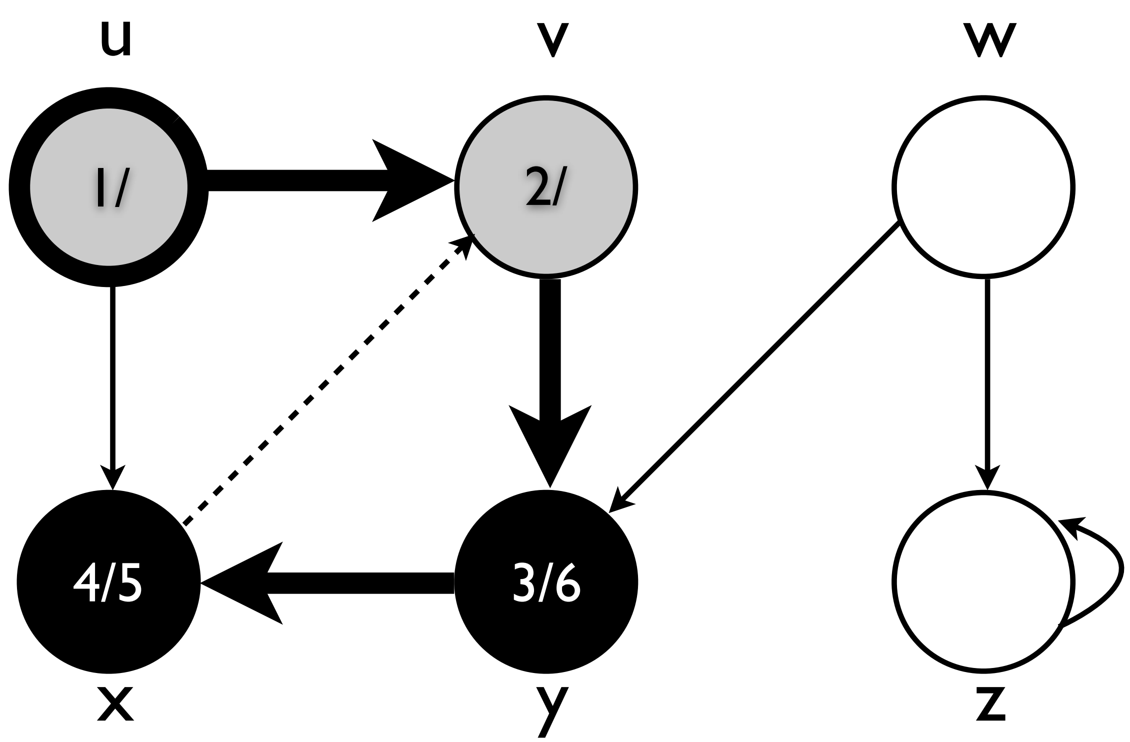
Busca em Profundidade



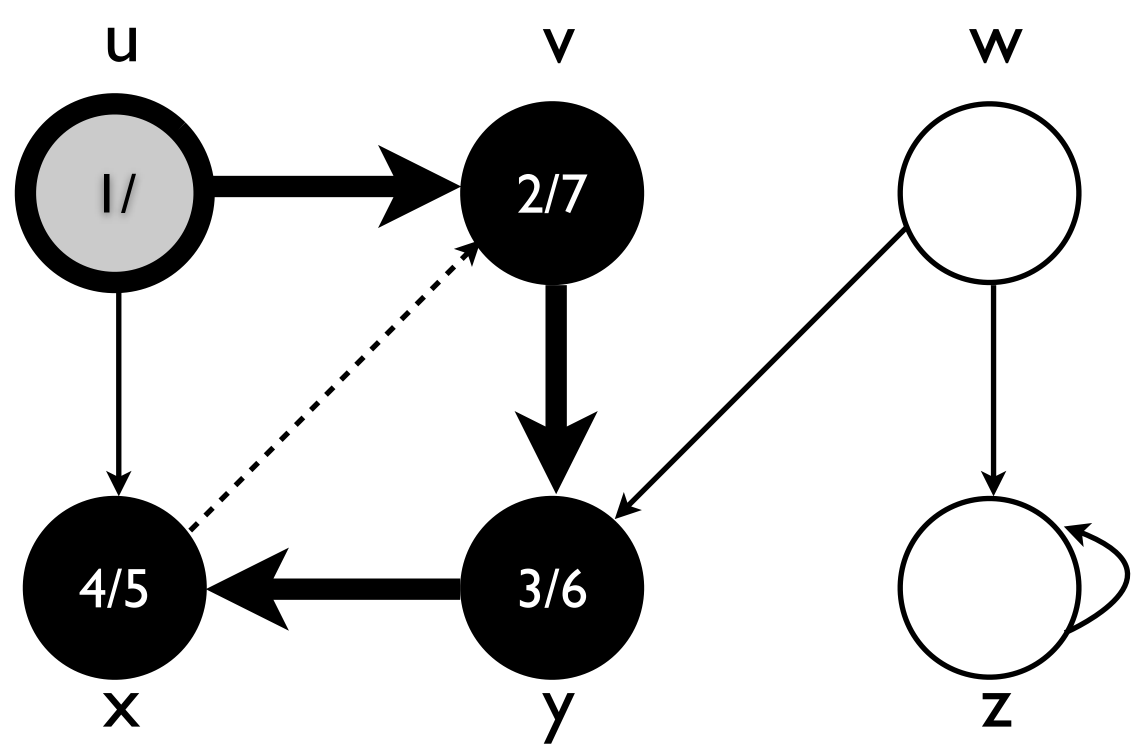
Busca em Profundidade



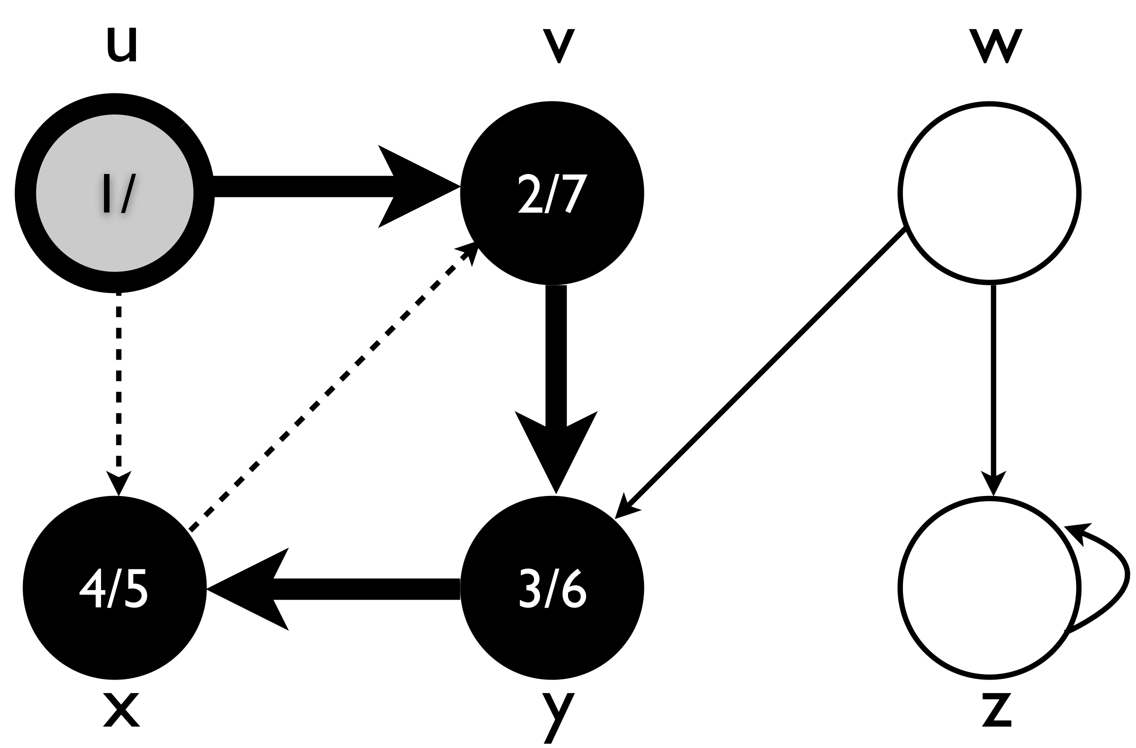
Busca em Profundidade



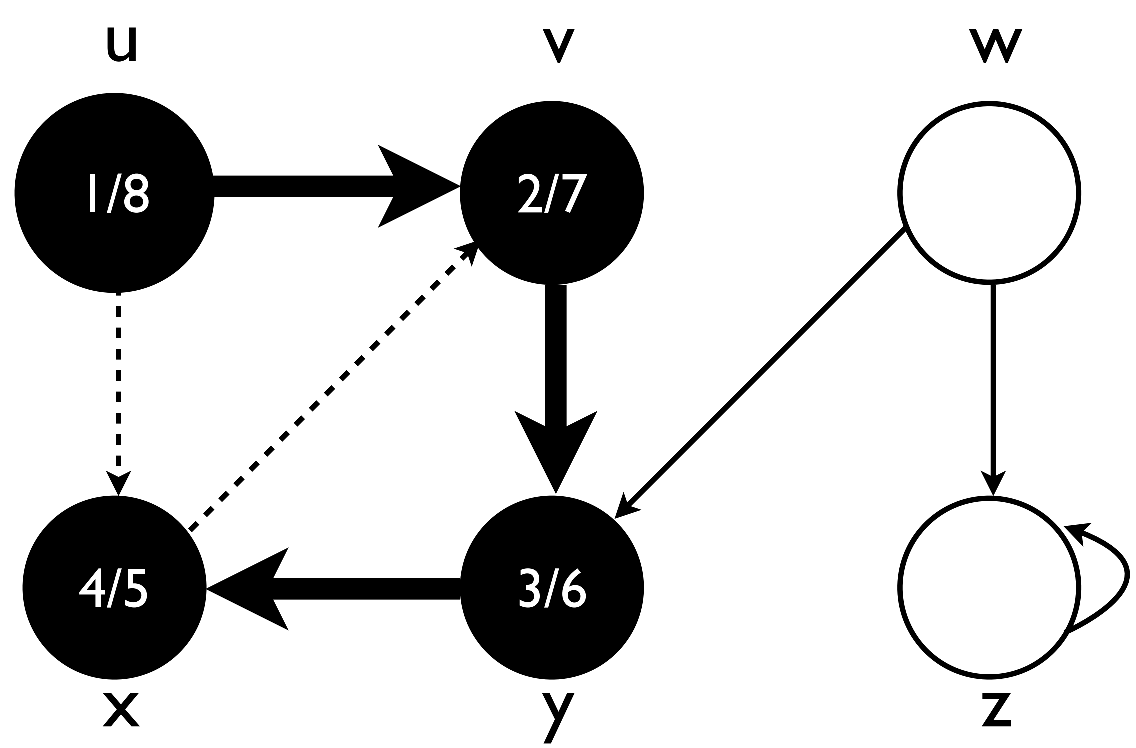
Busca em Profundidade



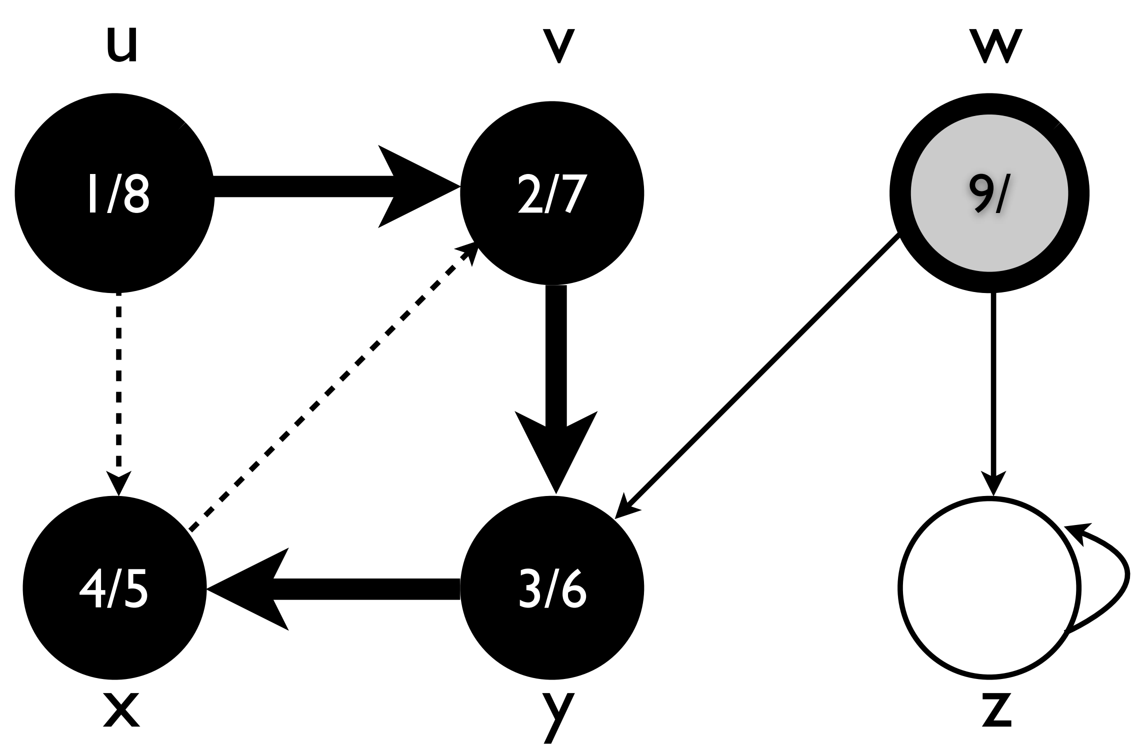
Busca em Profundidade



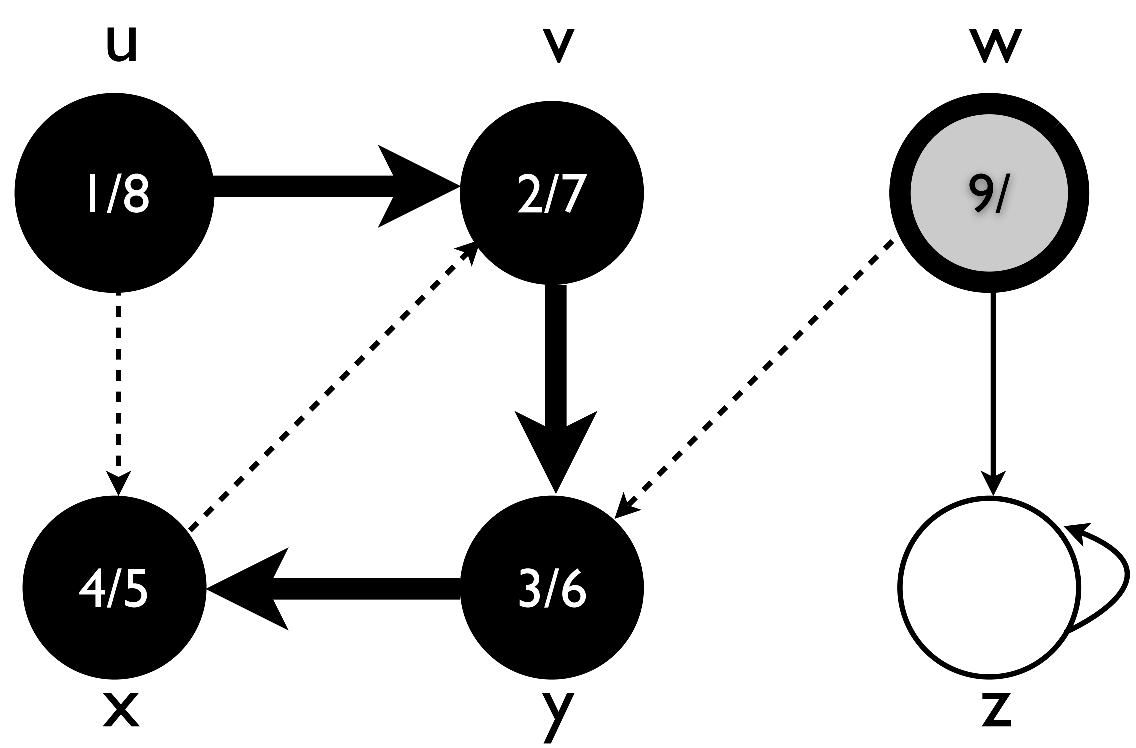
Busca em Profundidade



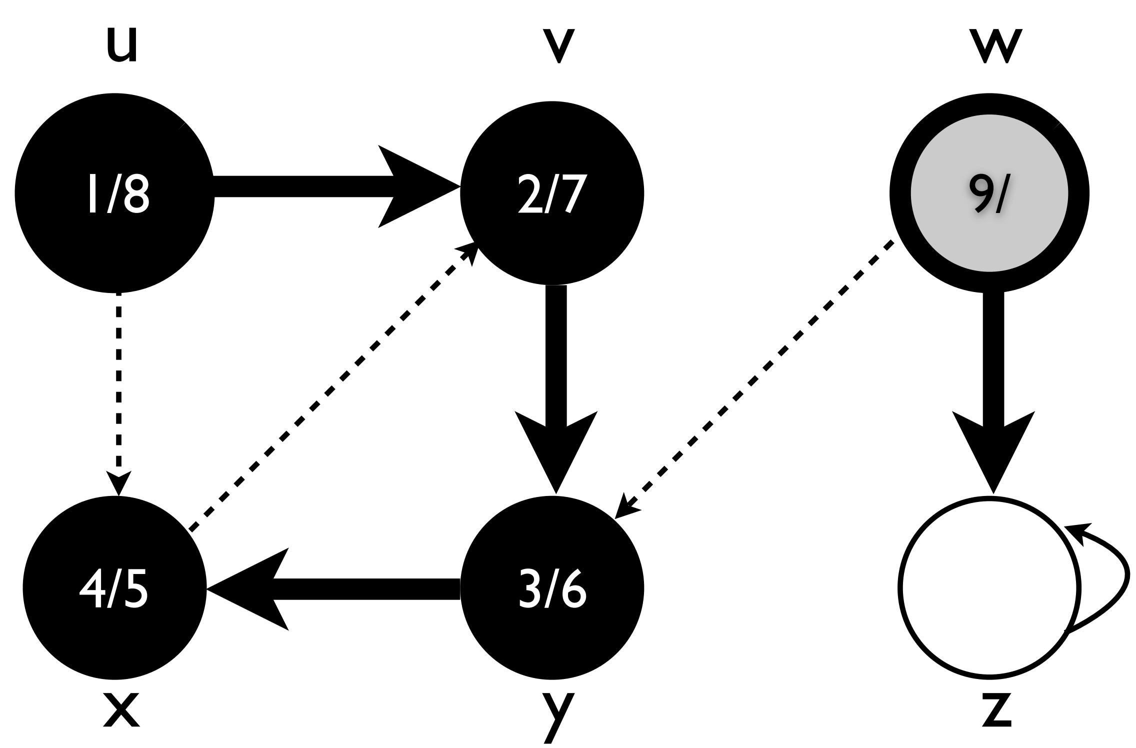
Busca em Profundidade



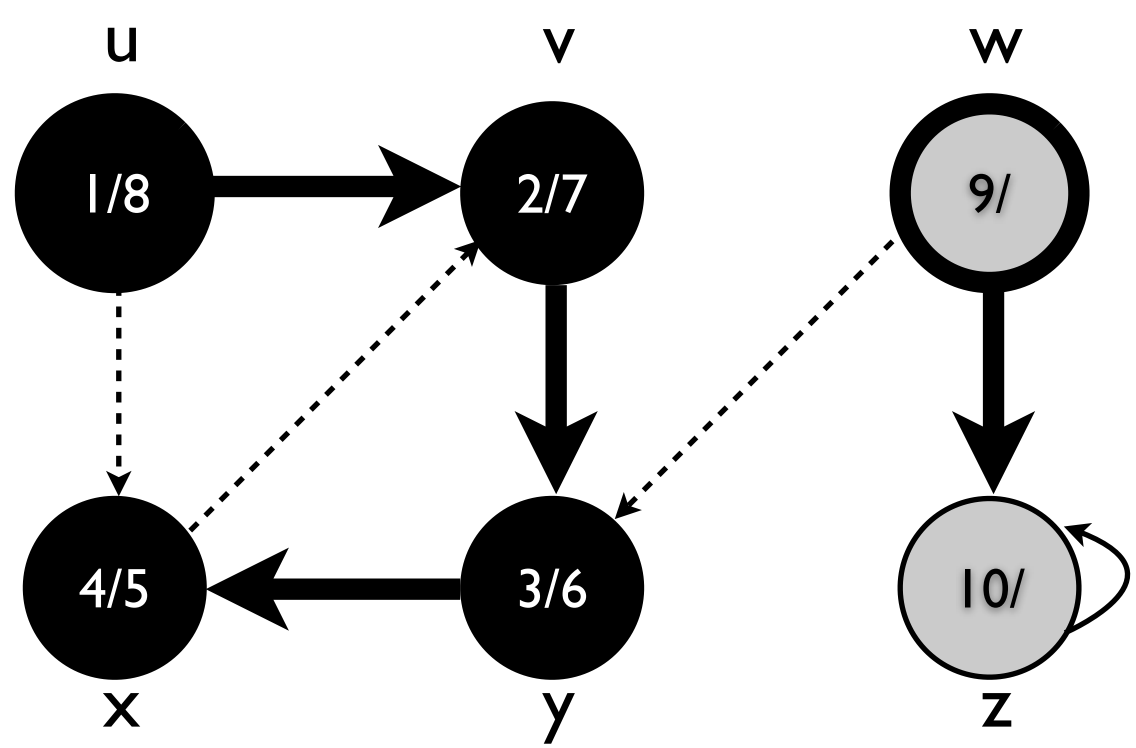
Busca em Profundidade



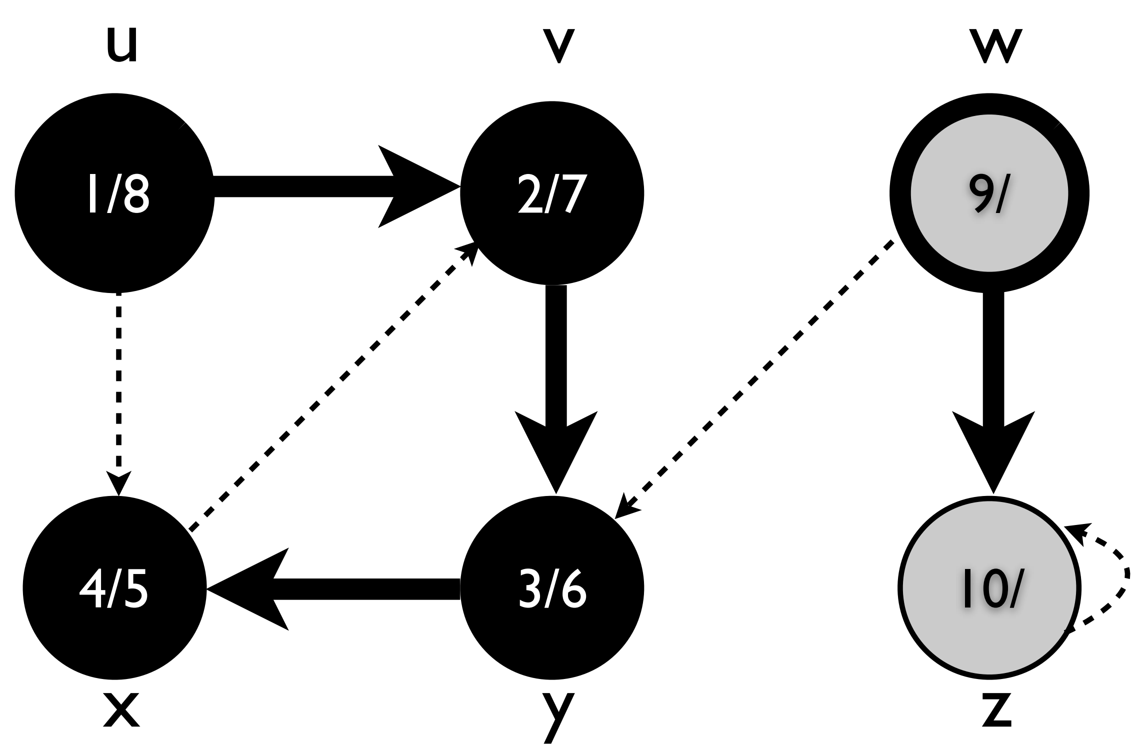
Busca em Profundidade



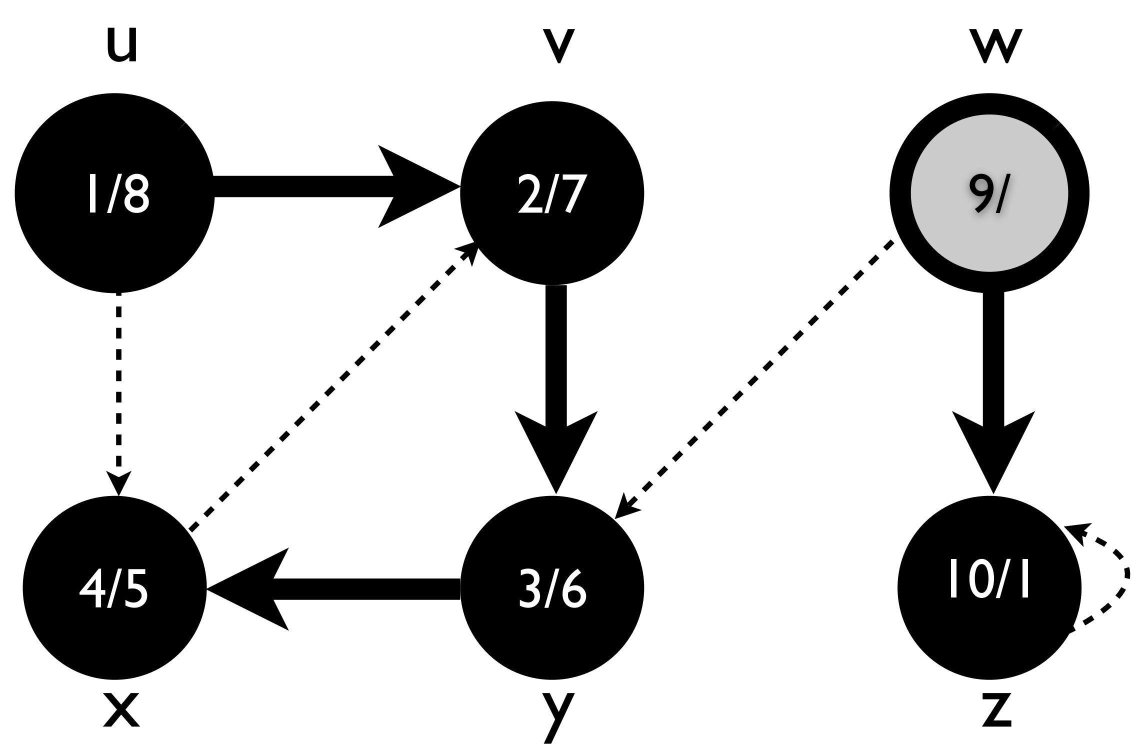
Busca em Profundidade



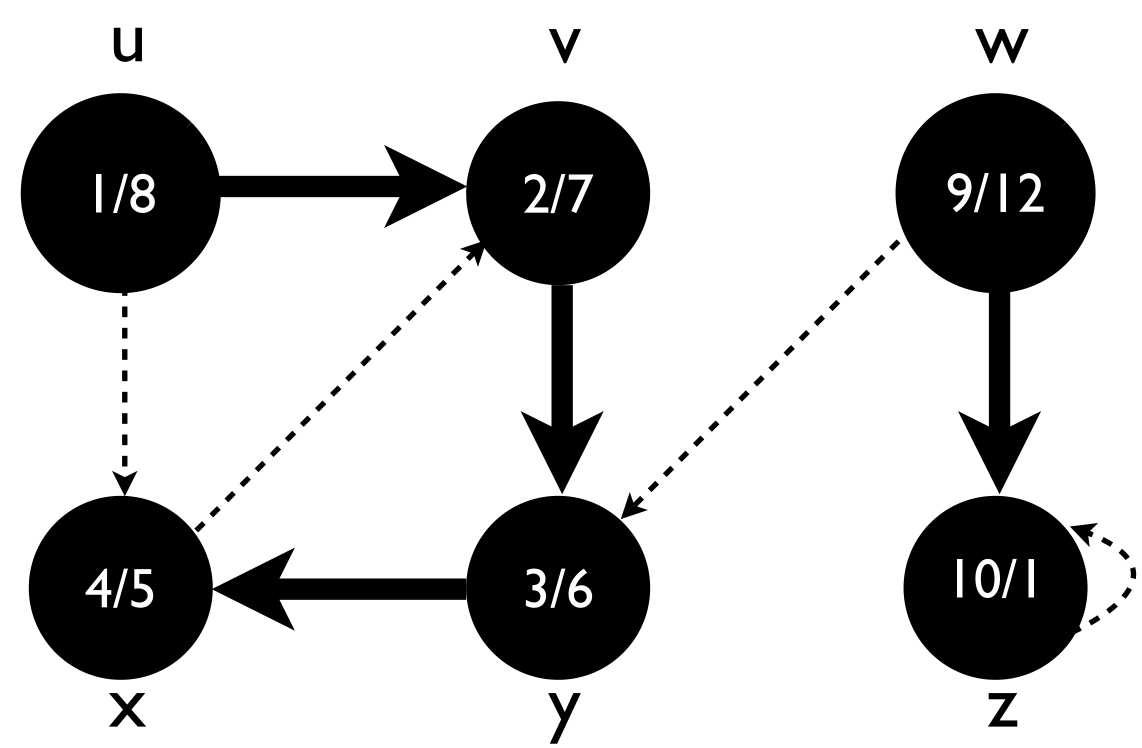
Busca em Profundidade



Busca em Profundidade



Busca em Profundidade



Algoritmo e Complexidade

Busca em Profundidade

```
BuscaProf(G)
    Para cada vértice u em V[G]
        cor_u = Branco; pai_u = NULL
    tempo = 0
    Para cada vértice u em V[G]
        Se(cor_u == Branco)
            Visitar(u)
```

Busca em Profundidade

1. **BuscaProf(G)**
2. **Para cada** vértice *u* em V[G]
3. cor_u = Branco; pai_u = NULL
4. tempo = 0
5. **Para cada** vértice *u* em V[G]
6. **Se**(cor_u == Branco)
7. Visitar(u)

O(|V|)

Busca em Profundidade

```
1. BuscaProf(G)
2.     Para cada vértice u em V[G]                         O(|V|)
3.         cor_u = Branco; pai_u = NULL
4.     tempo = 0
5.     Para cada vértice u em V[G]
6.         Se(cor_u == Branco)                             ???
7.             Visitar(u)
```

Busca em Profundidade

Executado vezes $|Adj[u]|$

1. **Visitar(u)**
2. cor_u = cinza; tempo = tempo + 1; d_u = tempo;
3. **Para cada** *v* em Adj[*u*]
4. **Se** (cor_v == Branco)
5. $pai_v = u$
6. **Visitar**(v)
7. cor_u = Preto
8. f_u = tempo + 1; tempo = tempo + 1;

Busca em Profundidade

- Fazemos uso da Análise agregada
- *Visitar* é chamado apenas uma vez para cada vértice *v em V* mas apenas para vértices brancos
- Todas as chamadas a *Visitar* tem custo

$$\sum_{u \in V} |Adj[u]| = \Theta(E)$$

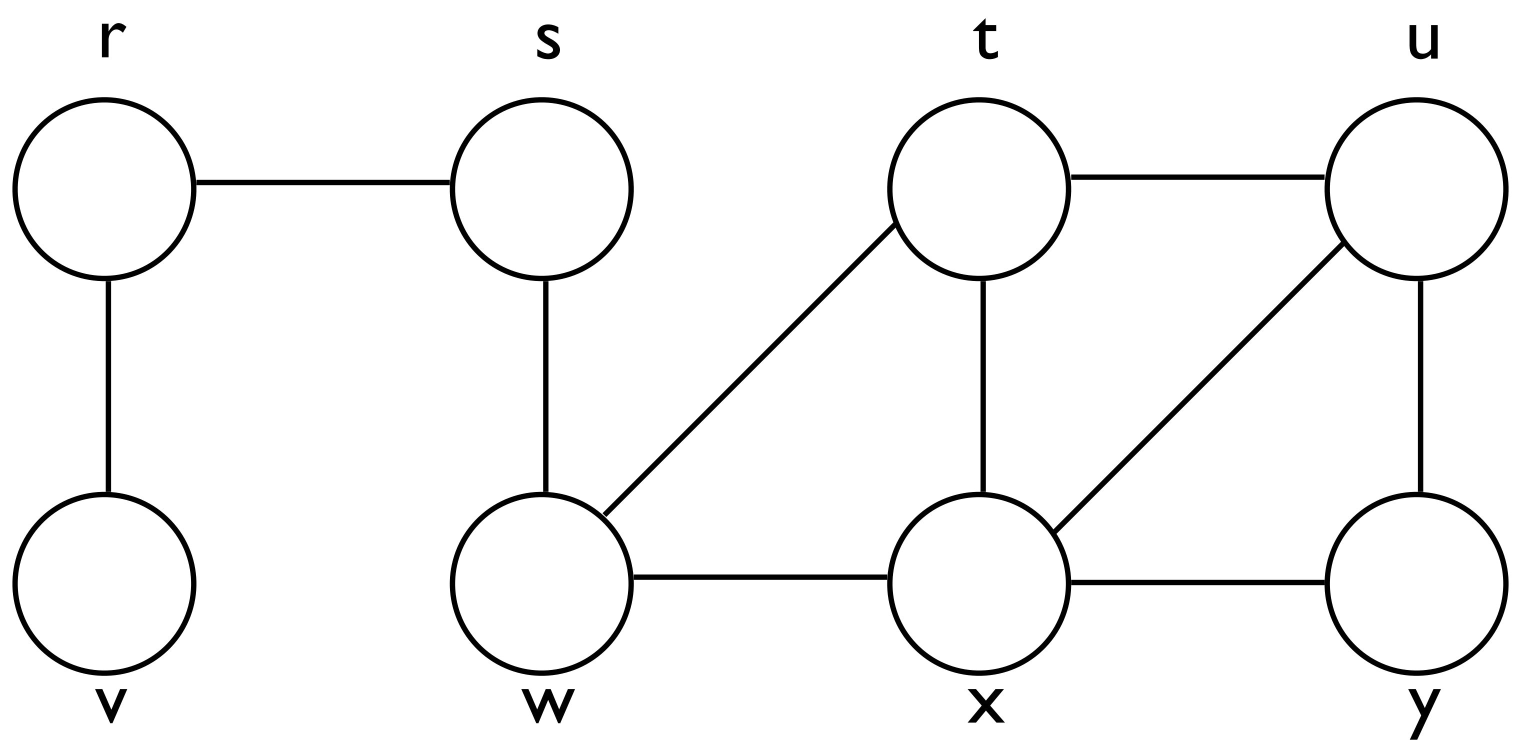
- Logo, custo do algoritmo é O(|V| + |E|)

Busca em Largura

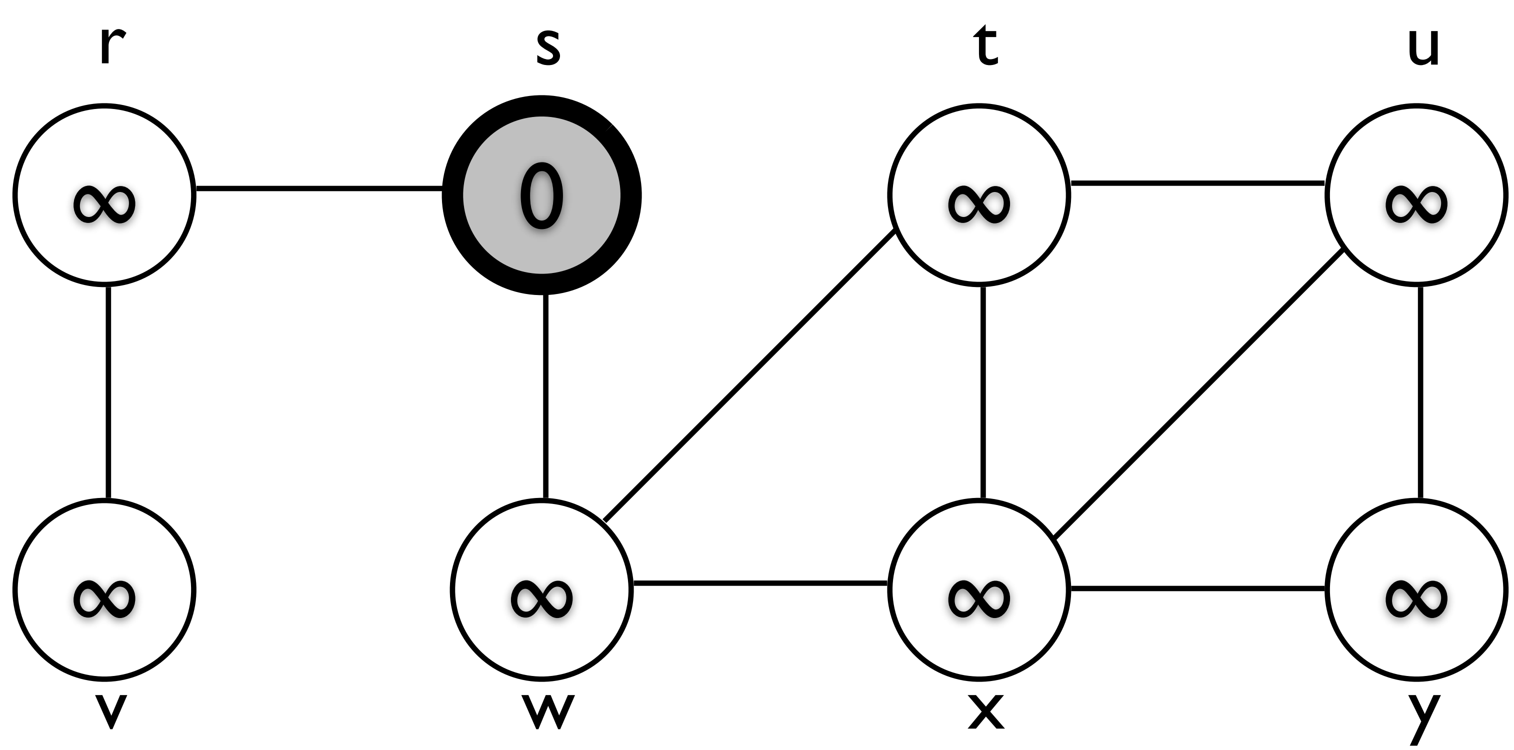
Busca em Largura

- Explicar busca em largura

Busca em Largura

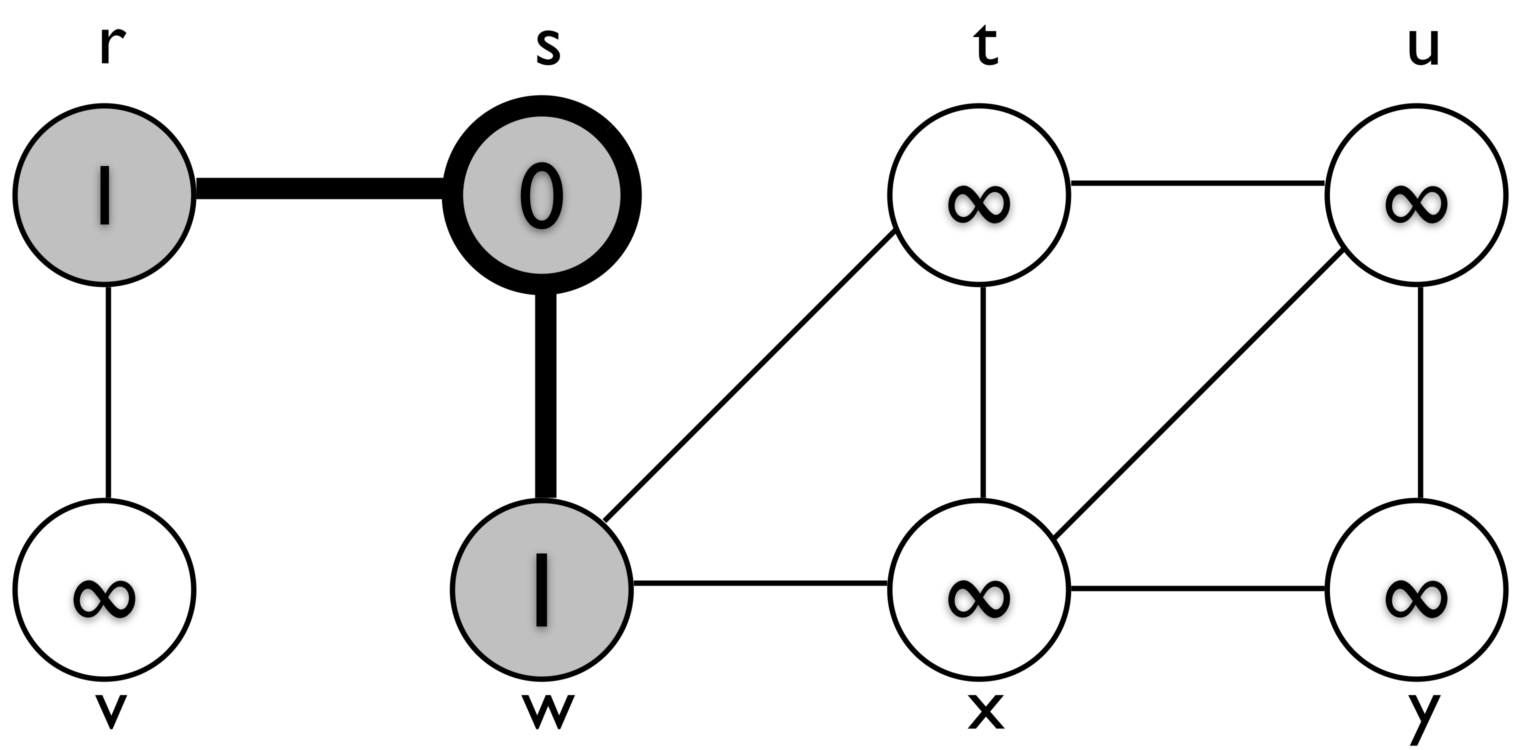


Busca em Largura



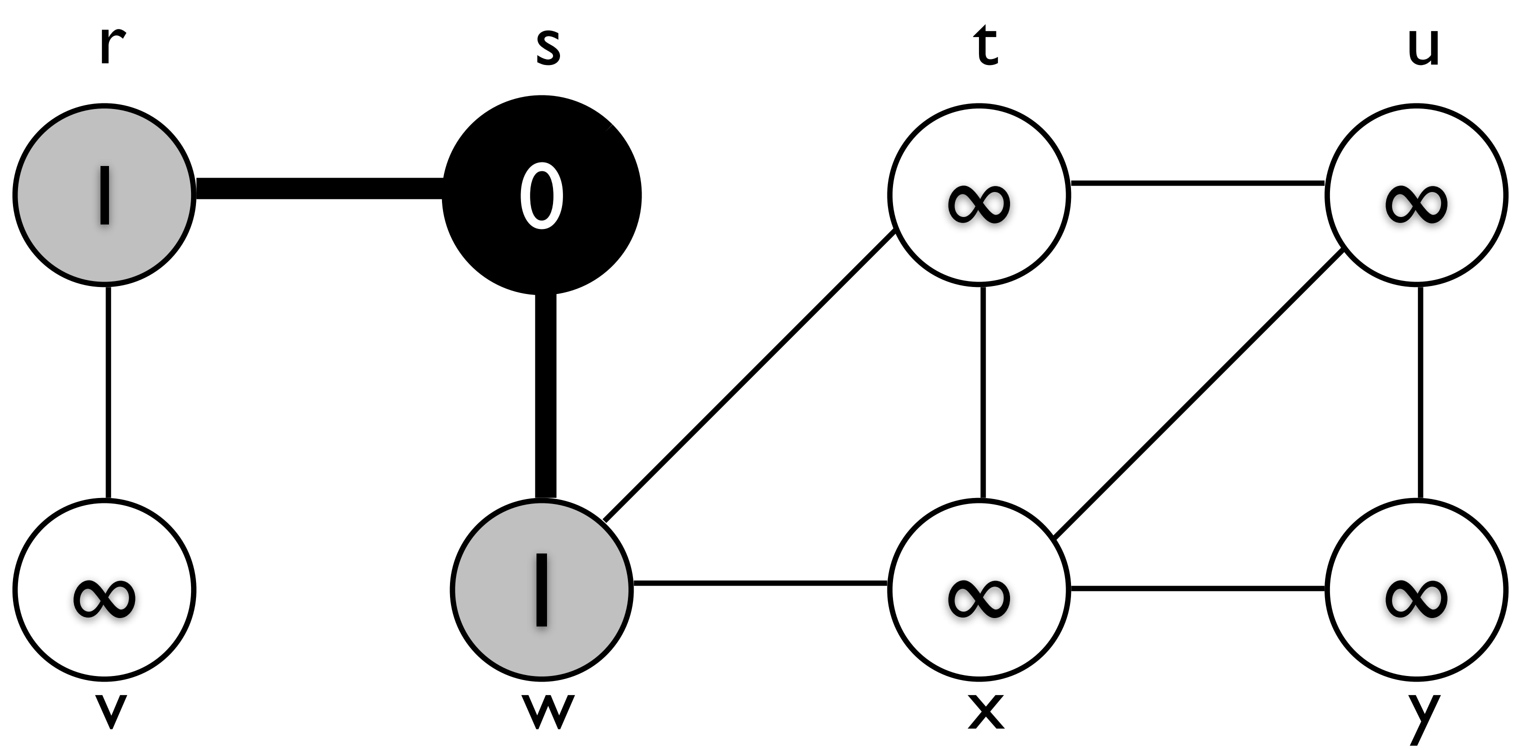
Fila Q	s
Nível	0

Busca em Largura



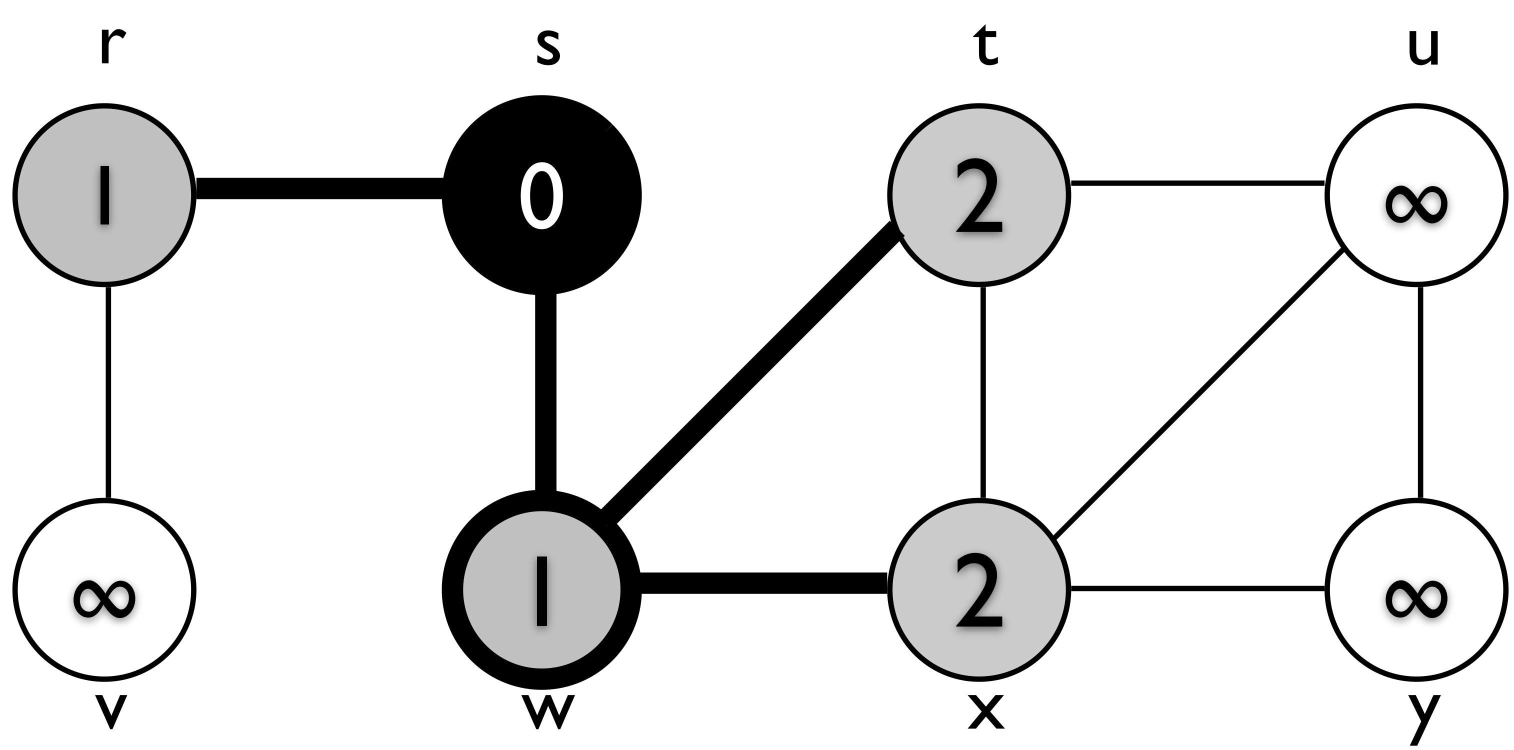
Fila Q	w	r
Nível	1	1

Busca em Largura



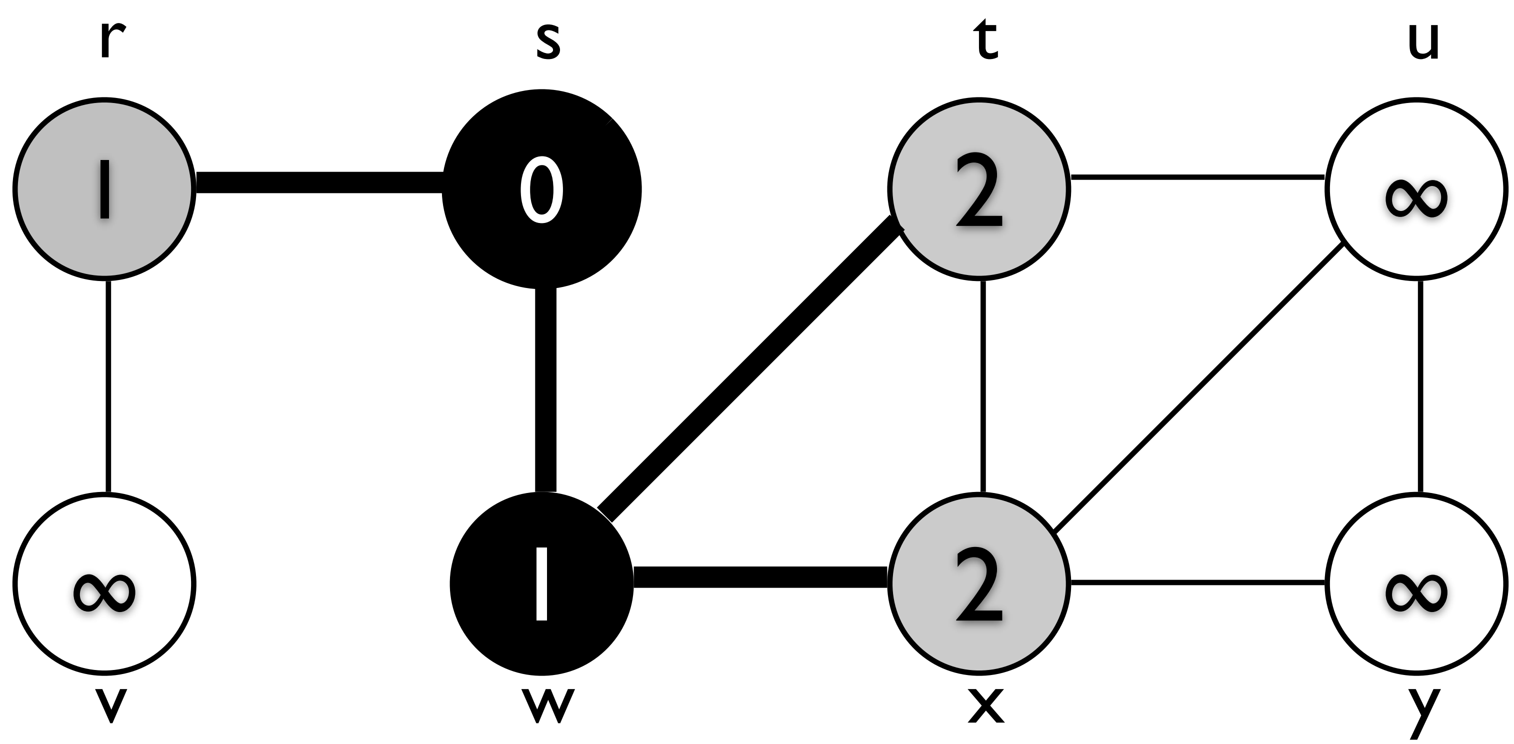
Fila Q	w	r
Nível	1	1

Busca em Largura



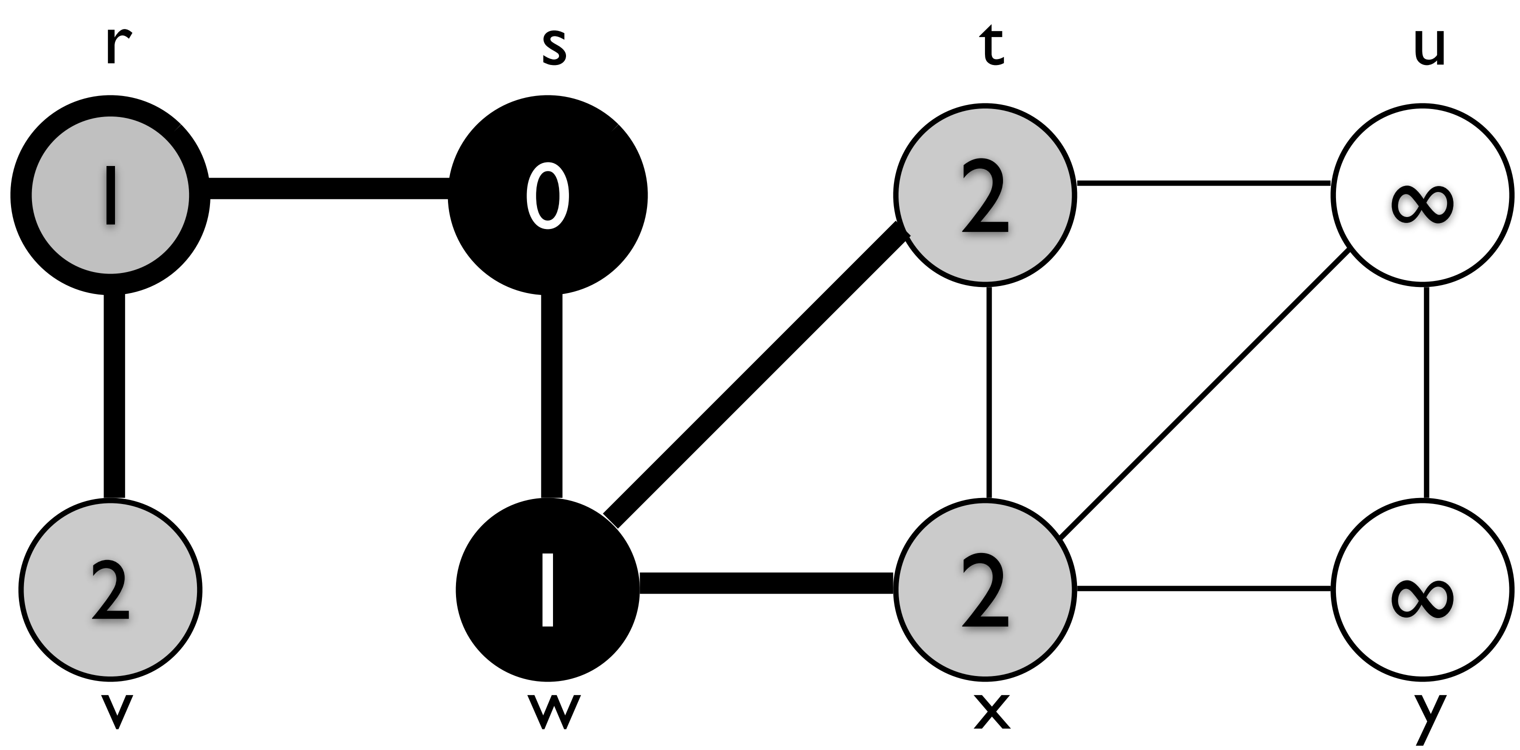
Fila Q	r	t	x
Nível	1	2	2

Busca em Largura



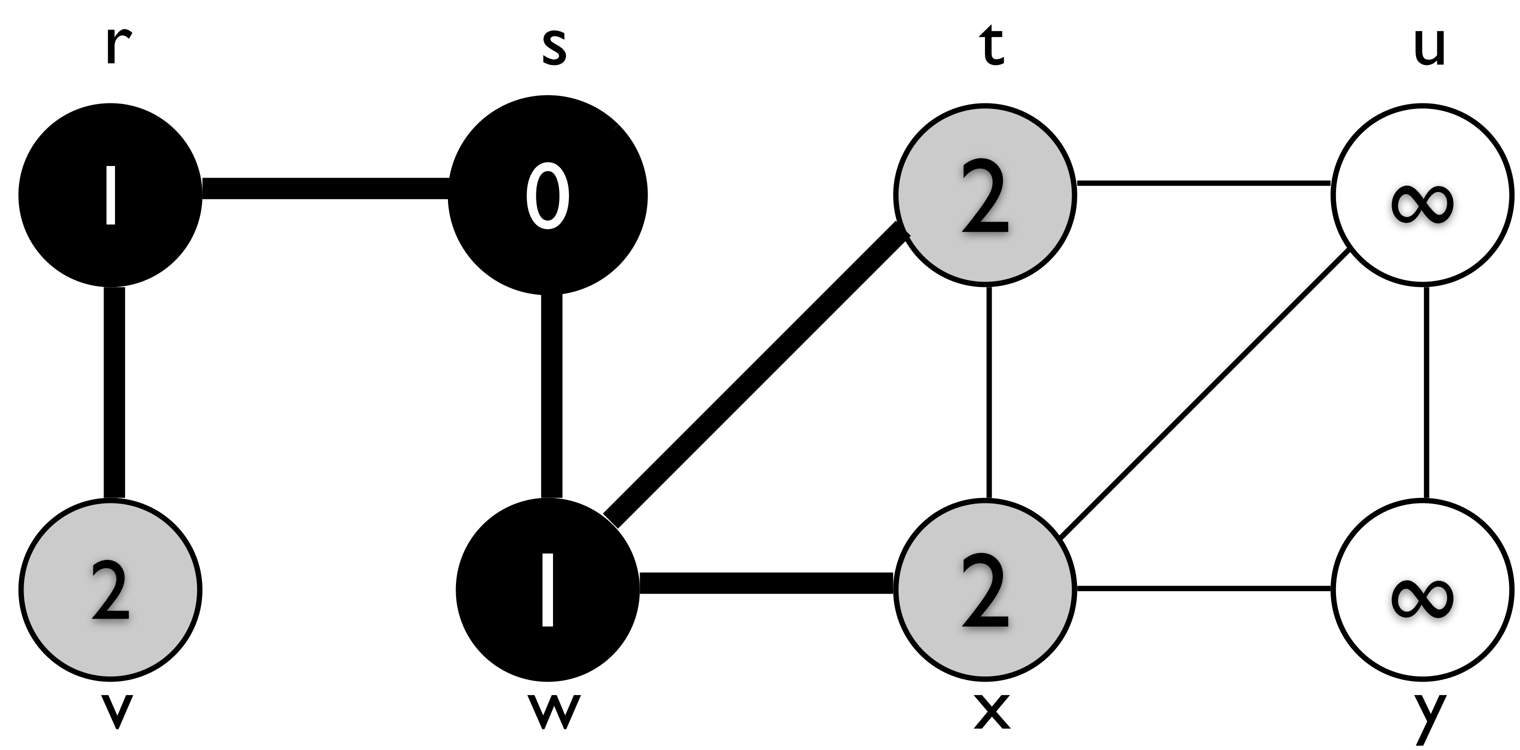
Fila Q	r	t	x
Nível	1	2	2

Busca em Largura



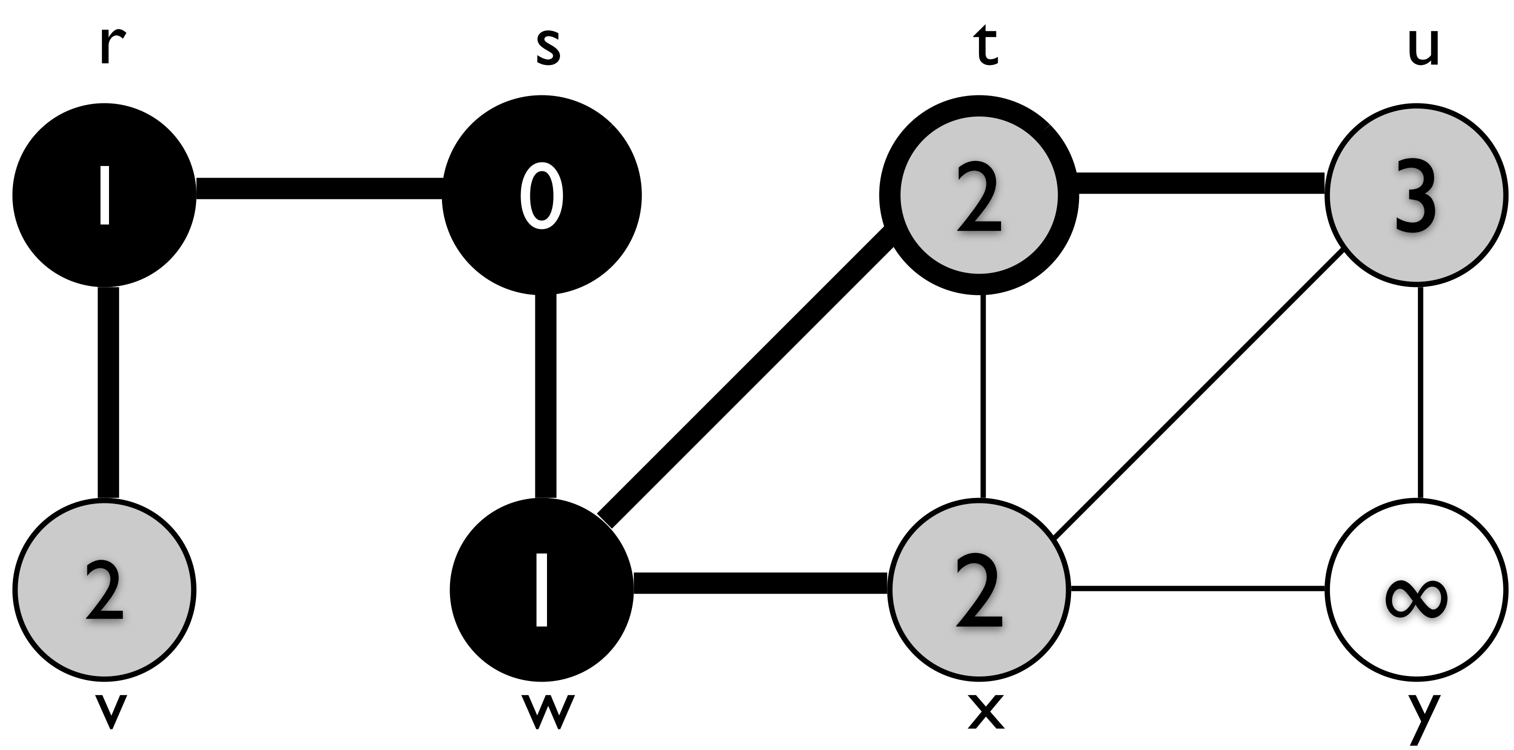
Fila Q	t	x	v
Nível	2	2	2

Busca em Largura



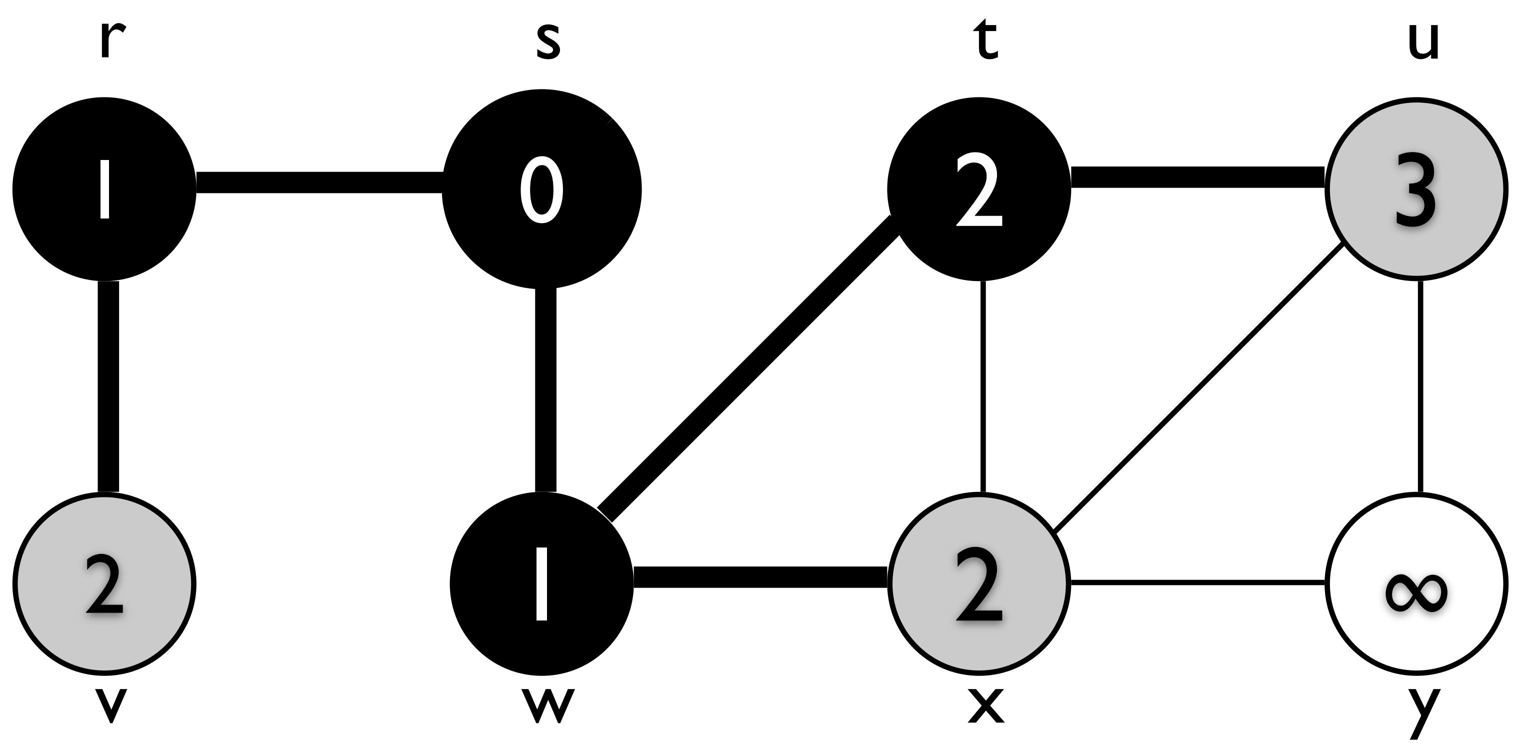
Fila Q	t	x	v
Nível	2	2	2

Busca em Largura



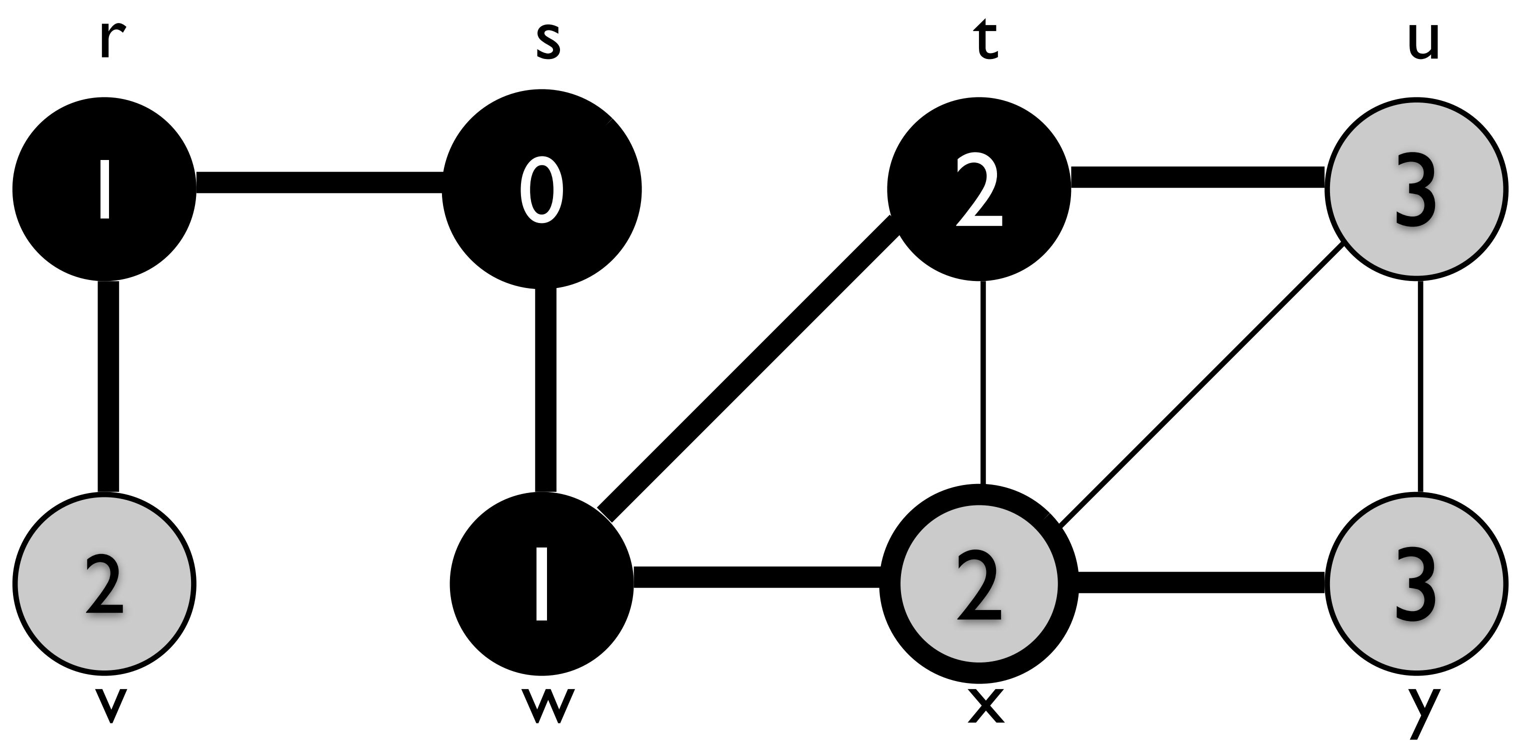
Fila Q	x	v	u
Nível	2	2	3

Busca em Largura



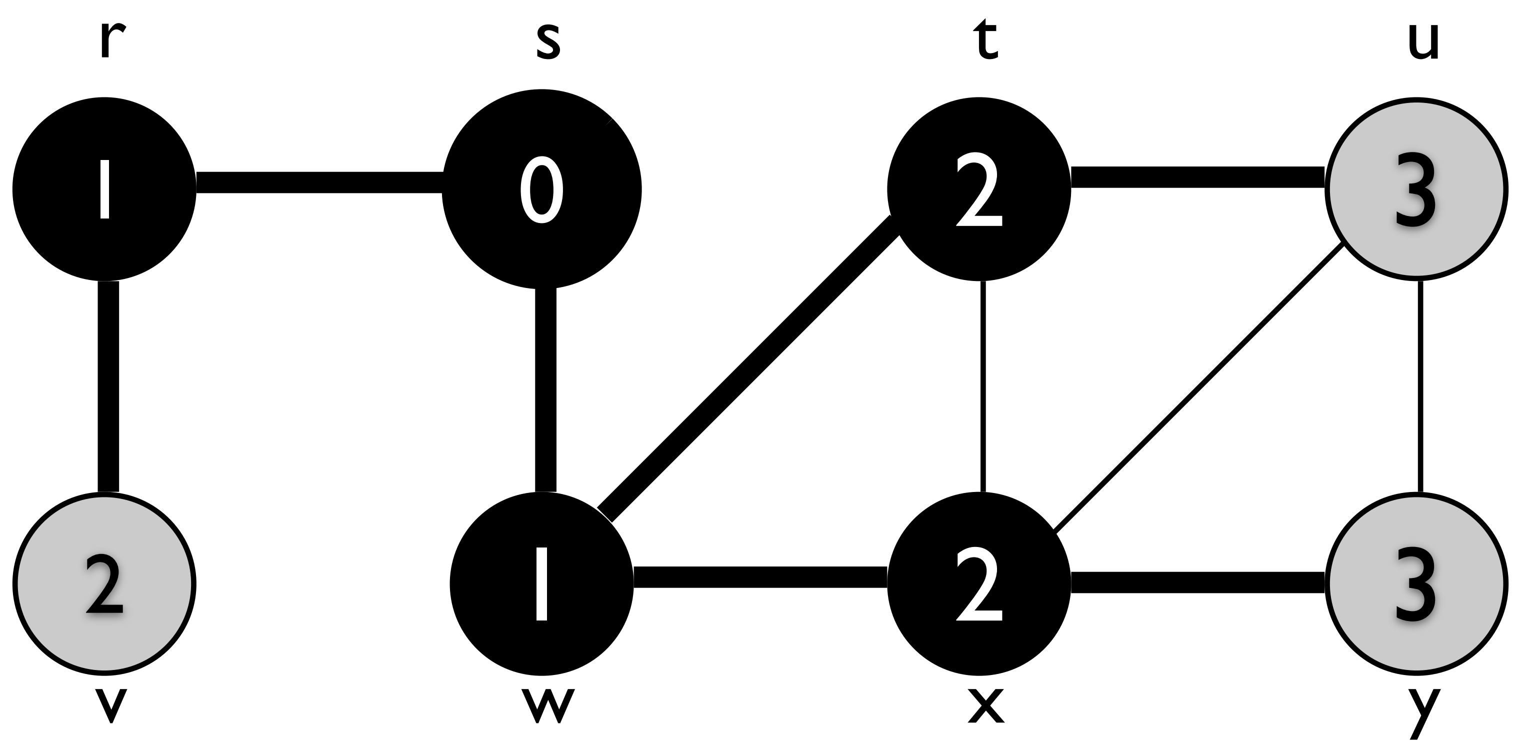
Fila Q	x	v	u
Nível	2	2	3

Busca em Largura



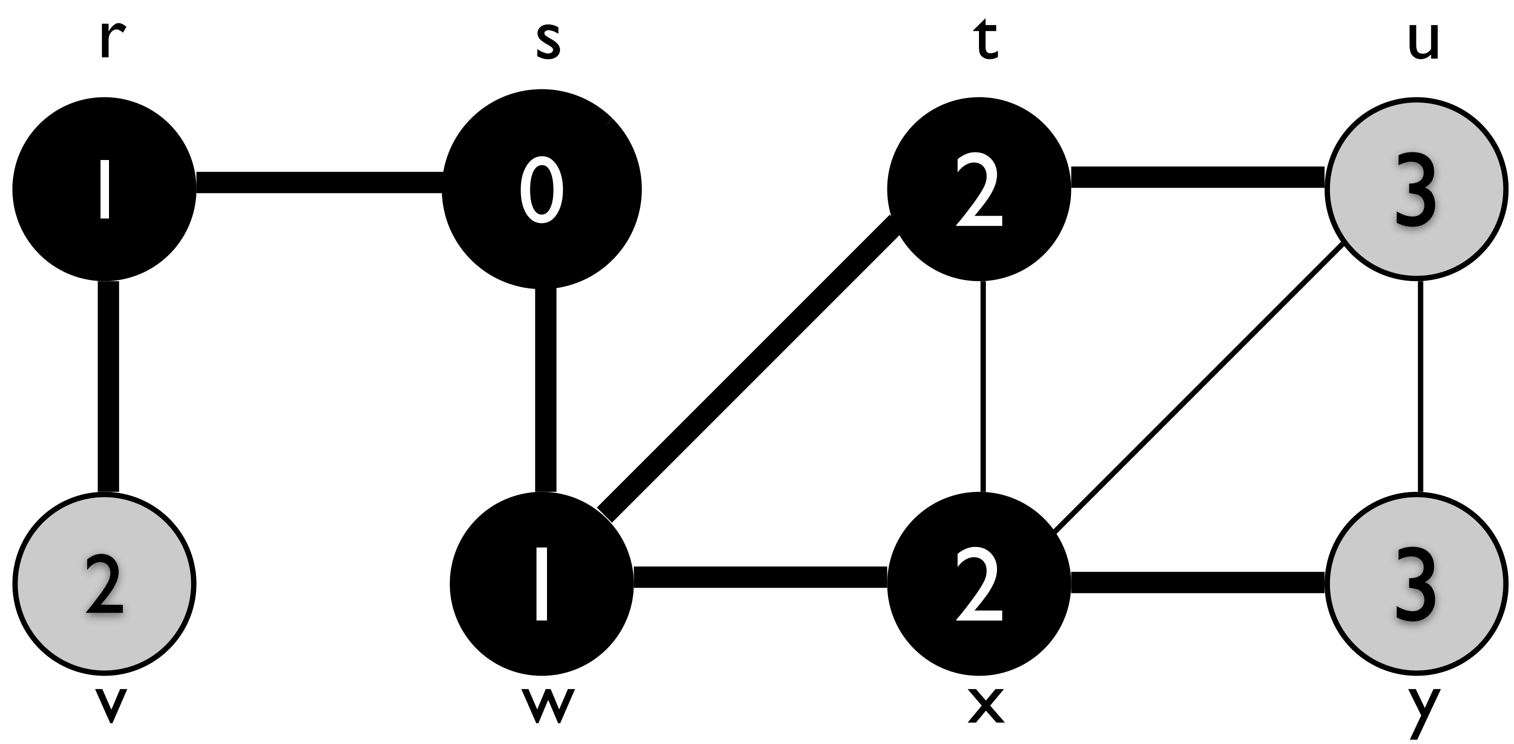
Fila Q	v	u	y
Nível	2	3	3

Busca em Largura



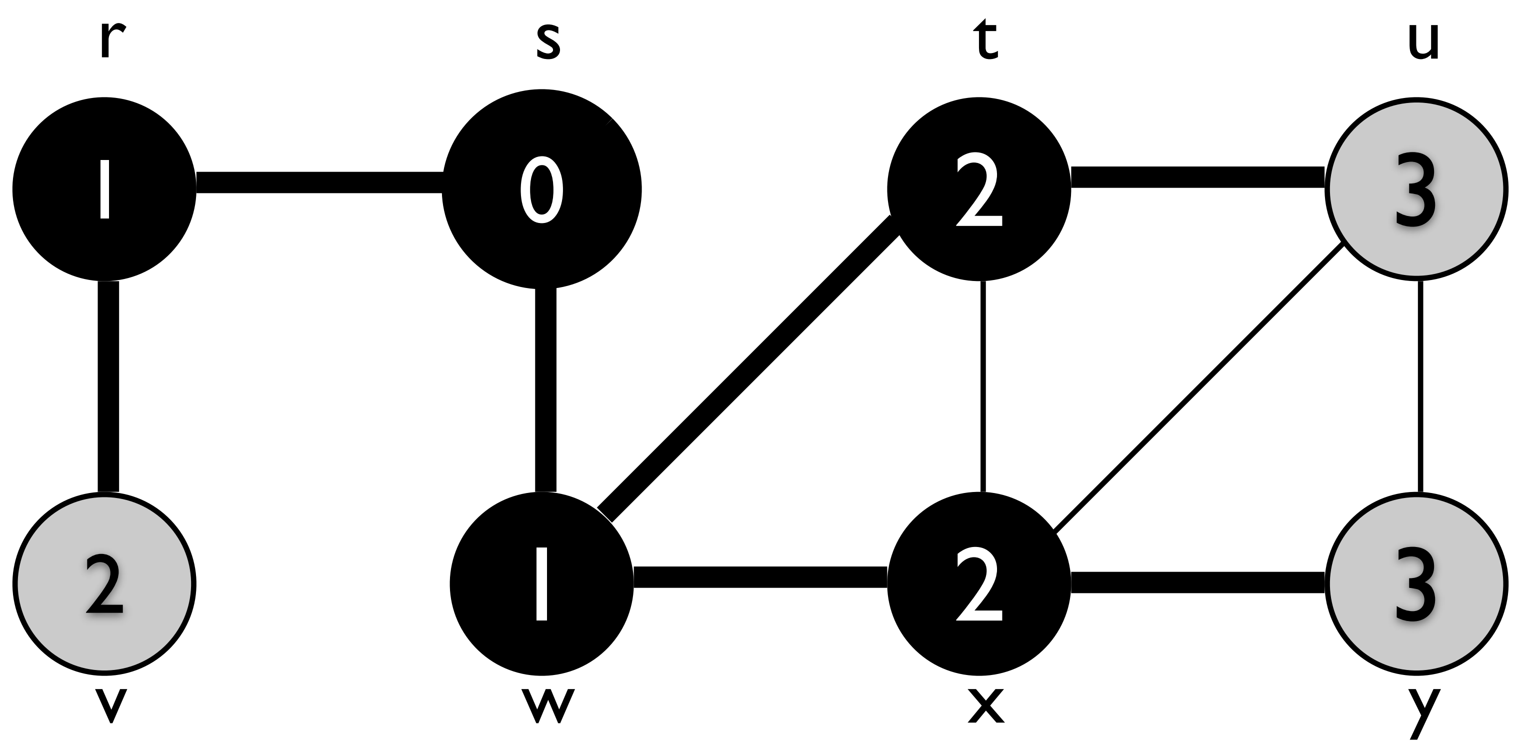
Fila Q	v	u	y
Nível	2	3	3

Busca em Largura



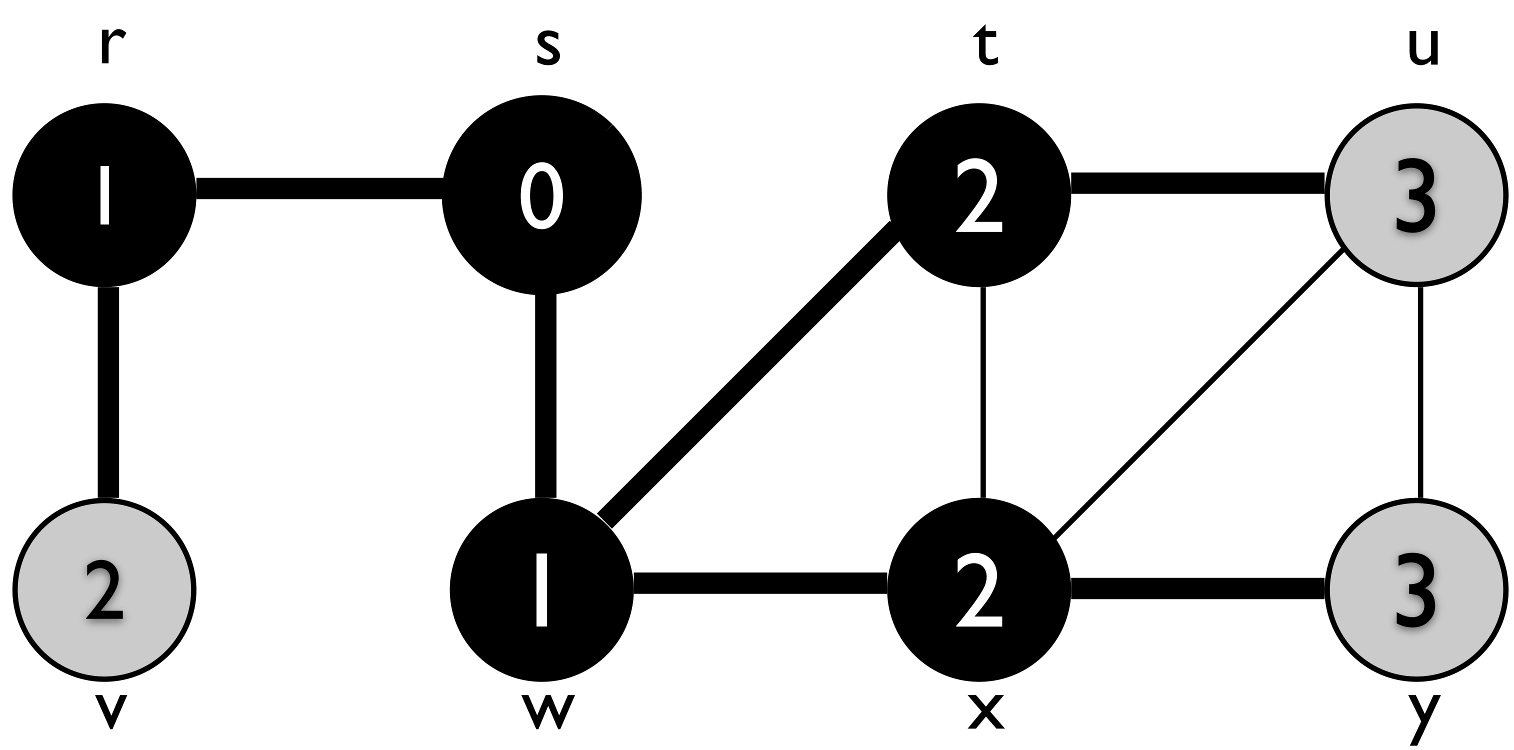
Fila Q	u	y
Nível	3	3

Busca em Largura



Fila Q	y
Nível	3

Busca em Largura



Fila Q	ø
Nível	–

Busca em Largura – Algoritmo

```
BuscaLarg(G)
    Para cada vértice u em V[G] - {S}
        cor_u = Branco; d_u = 0; pai_u = NULL
    cor_s = Cinza
    d_s = 0;  pai_s = NULL;
    Q = {}
    Enqueue(Q,s)
```

Busca em Largura – Algoritmo

```
(...)
while(Q != {})
    u = DeQueue(Q)
    foreach v = Adj[u]
        if(cor_v BRANCO)
            cor_v = CINZA; d_v = d_u + 1; pai_v = u;
            Enqueue(Q,v)
    cor_u = PRETO
```

Aplicações de Buscas

Onde podemos usar a Busca em Profundidade?

Ordenação Topológica

Aplicação I

Busca em Profundidade

- Podemos usar Busca em Profundidade para Ordenar Topologicamente um grafo
- Ordenação topológica é uma ordenação linear de todos os seus vértices, tal que se se o grafo G (acíclico*) tem uma aresta (*u*, *v*), então *u* aparece antes de *v* na ordenação

* *Grafo cíclico = não é possível ordem linear*

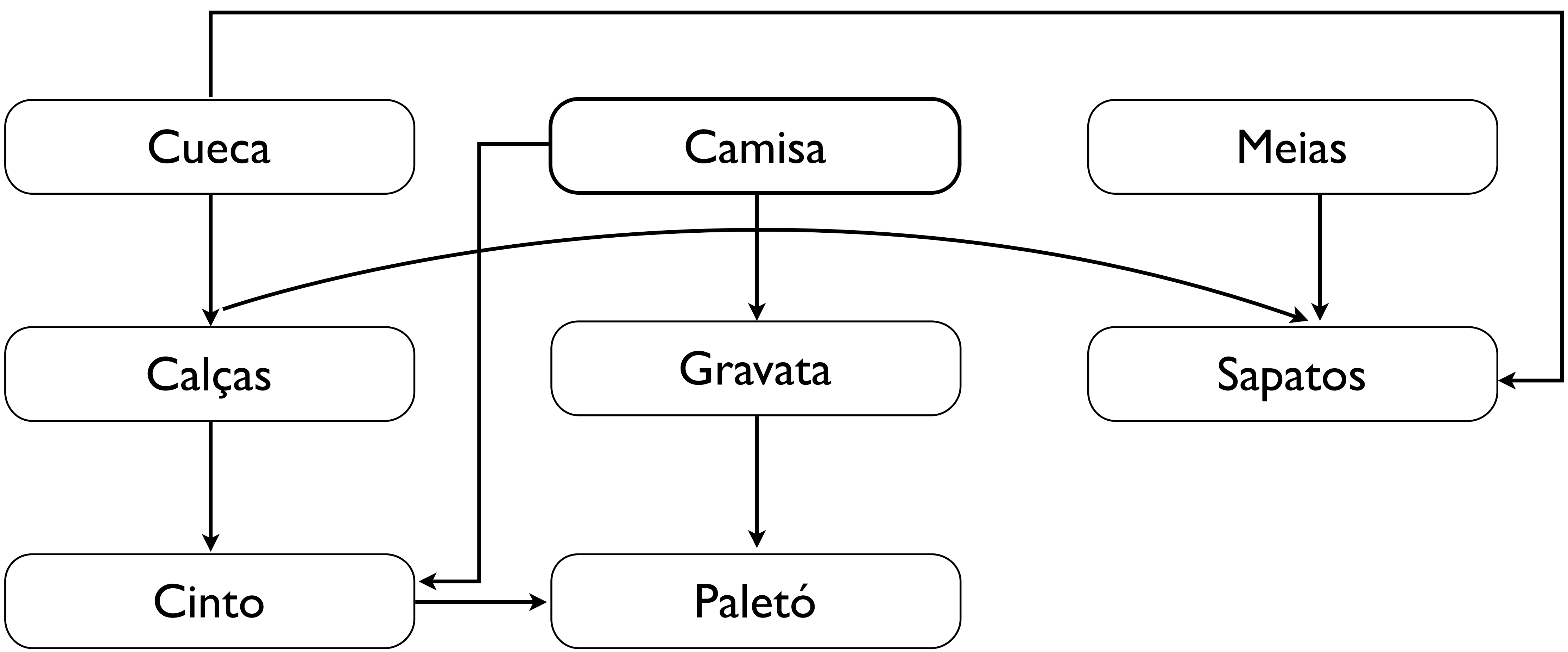
Busca em Profundidade

- Considere o problema
  - João quer definir a ordem de ações em que deve se preparar para uma reunião
  - Suas restrições são que ele deve vestir a cueca antes das calças e o cinto só pode ser colocado após vestir a calça
  - O paletó deve ser colocado após colocar os cintos
  - A camisa deve ser colocada antes do cinto

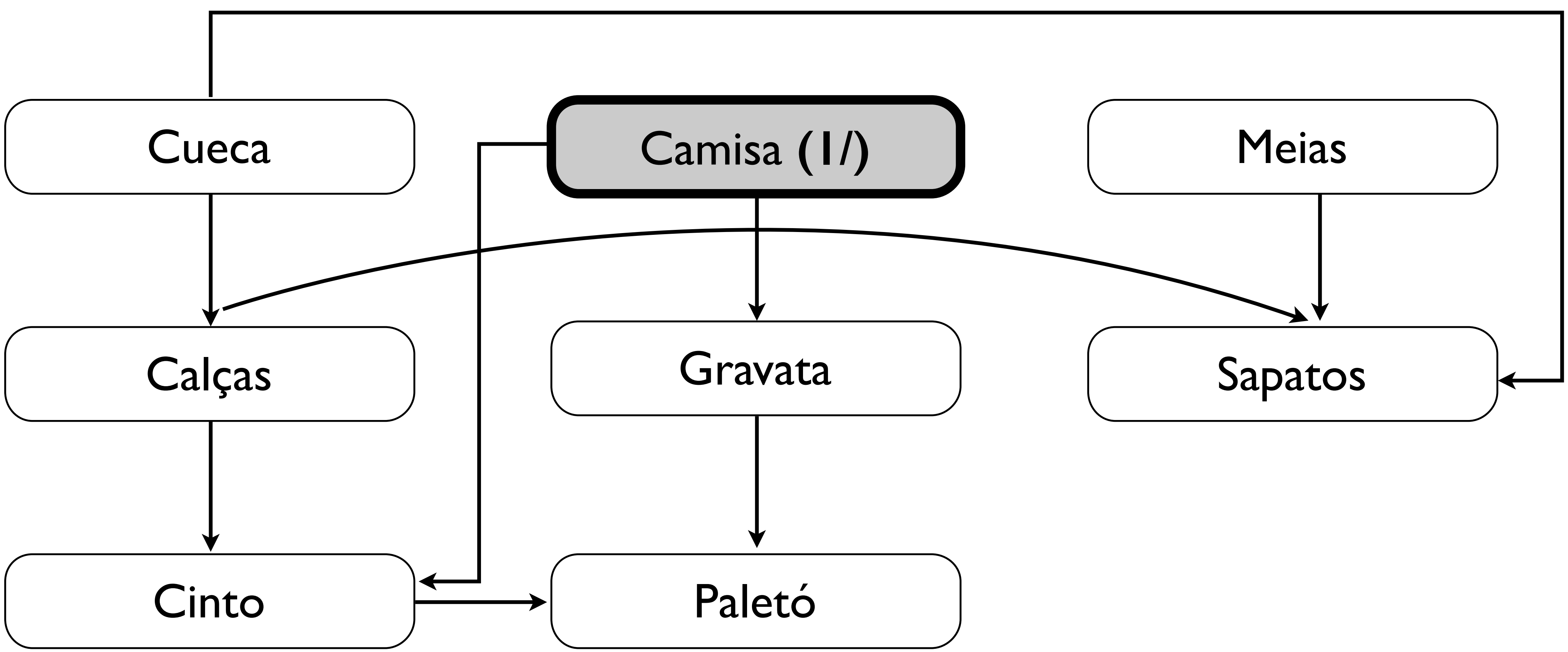
Busca em Profundidade

- Considere o problema
  - A gravata deve ser colocada depois da camisa e antes do paletó
  - As meias podem ser colocadas em qualquer ordem
  - As Cueca devem ser colocadas antes dos sapatos assim como as calças
- Como conseguir um plano de execução?

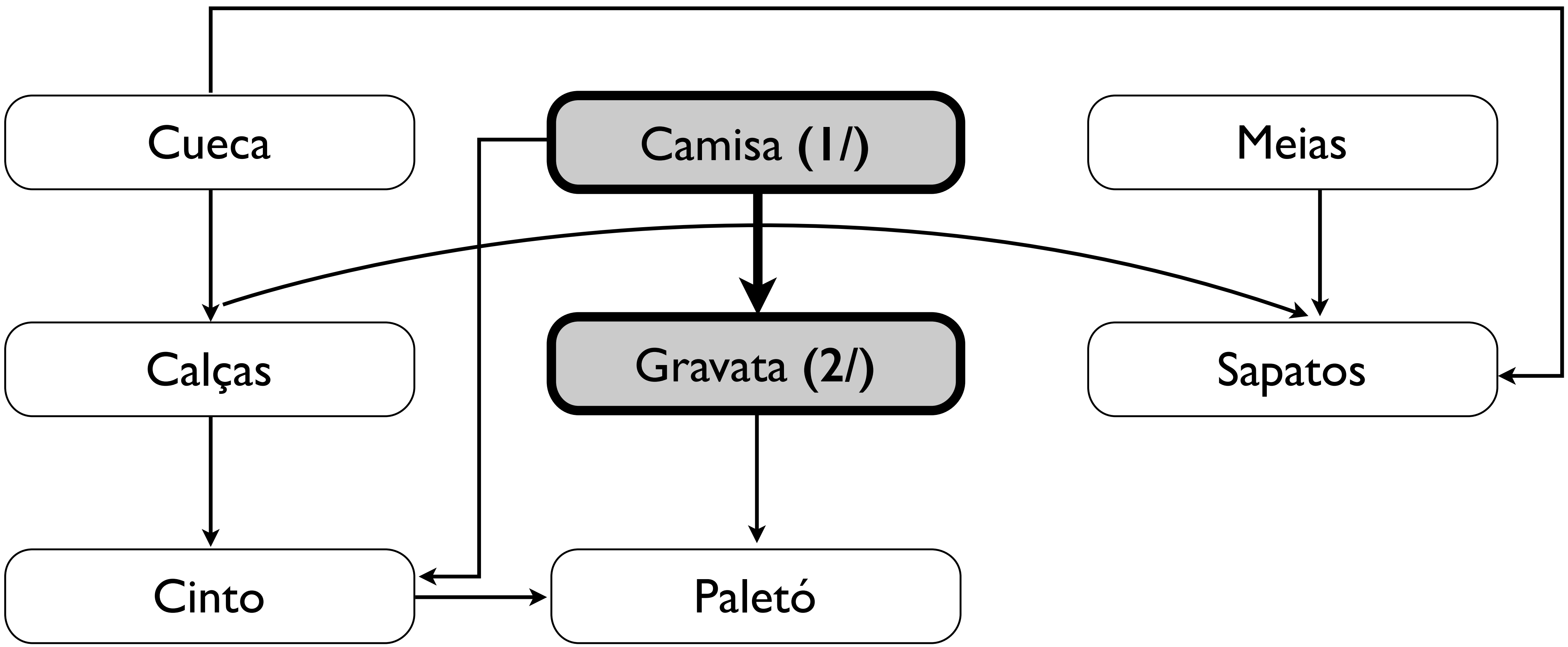
Ordenação Topológica



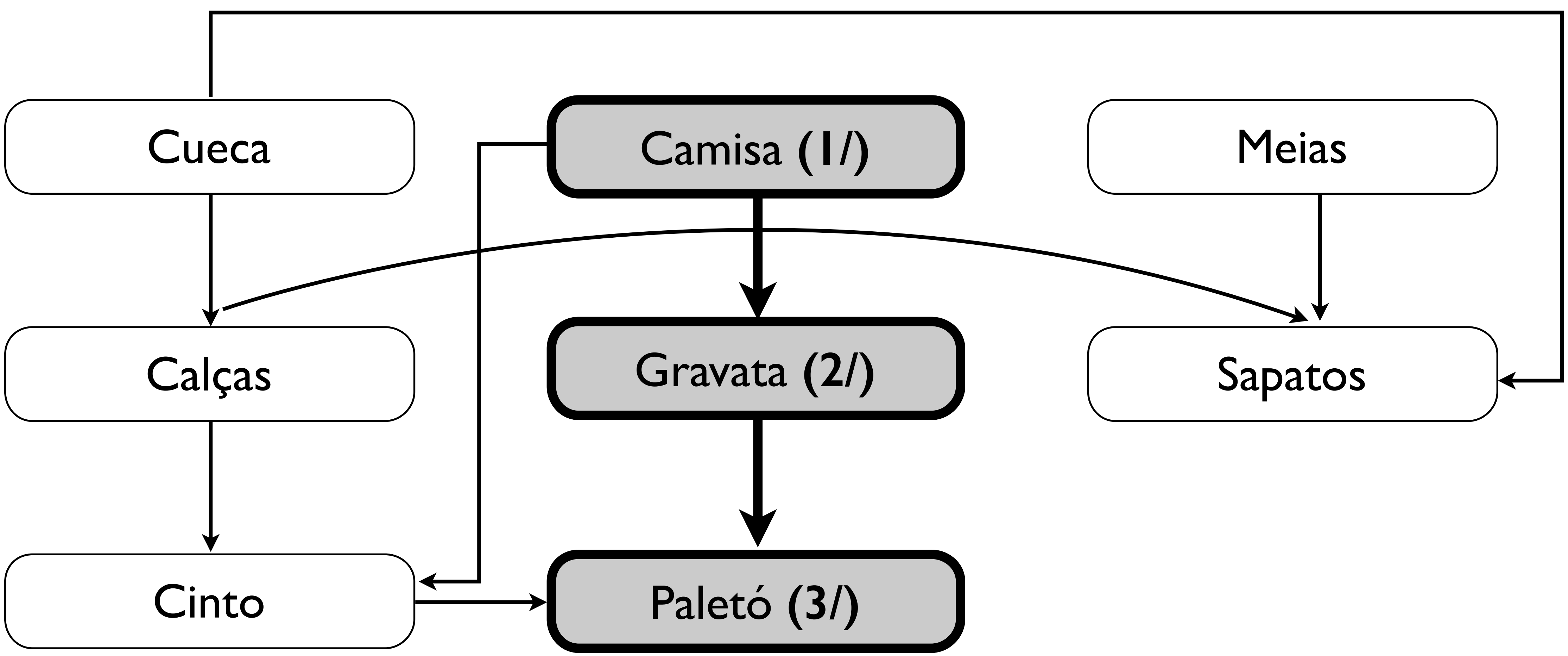
Ordenação Topológica



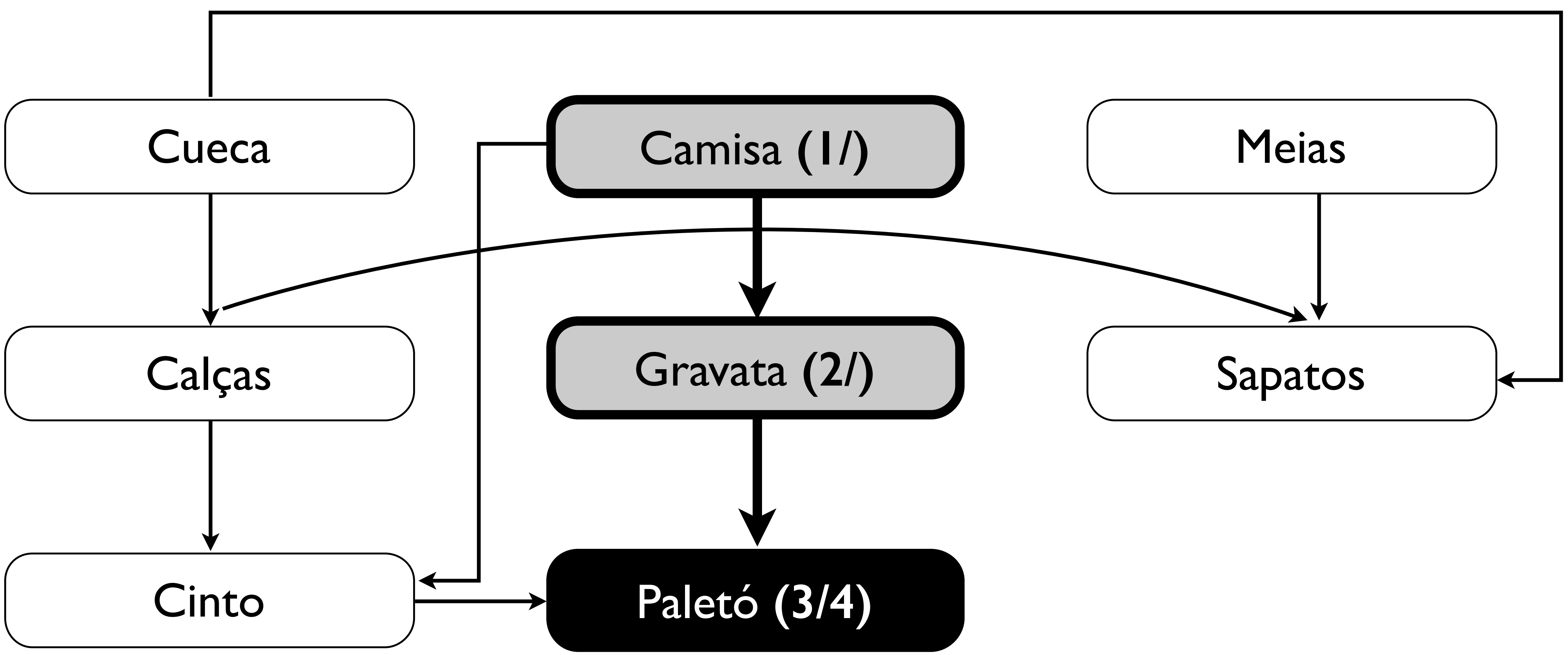
Ordenação Topológica



Ordenação Topológica

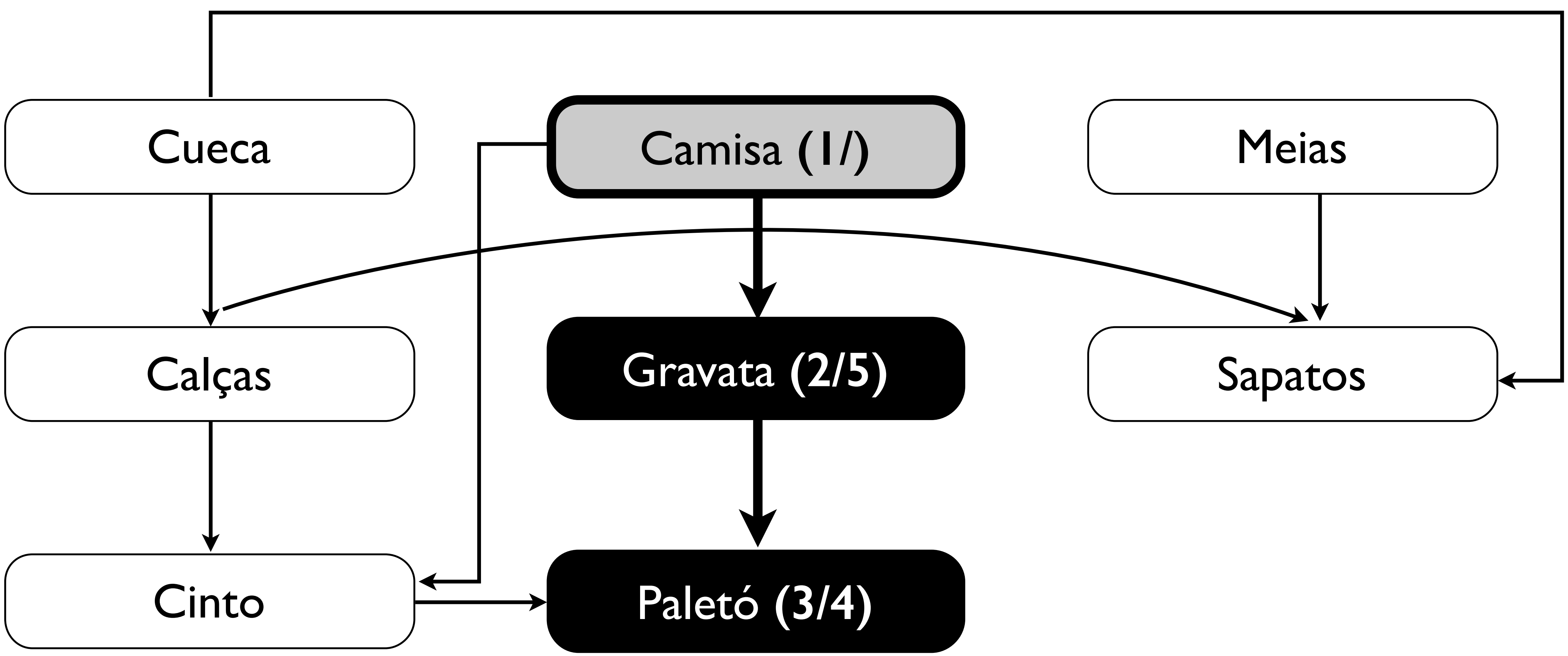


Ordenação Topológica



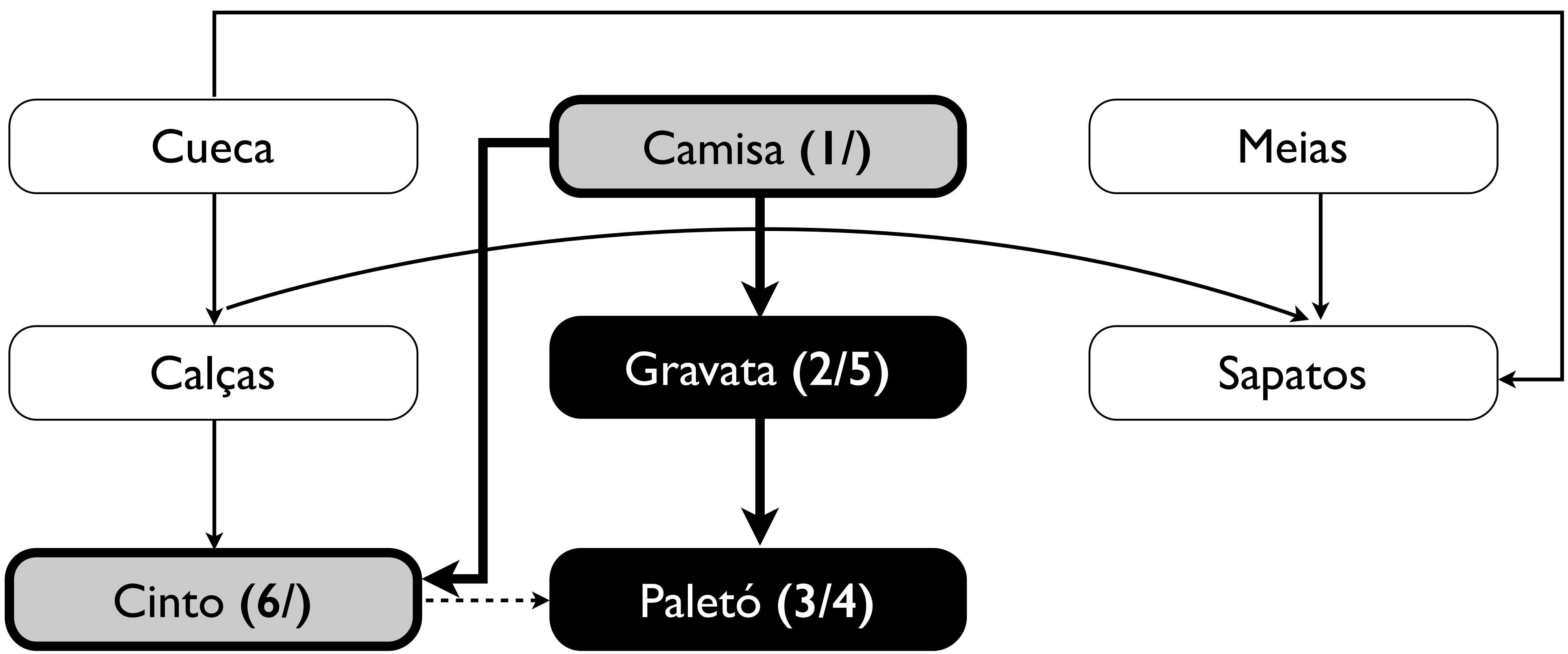
{Paletó}

Ordenação Topológica



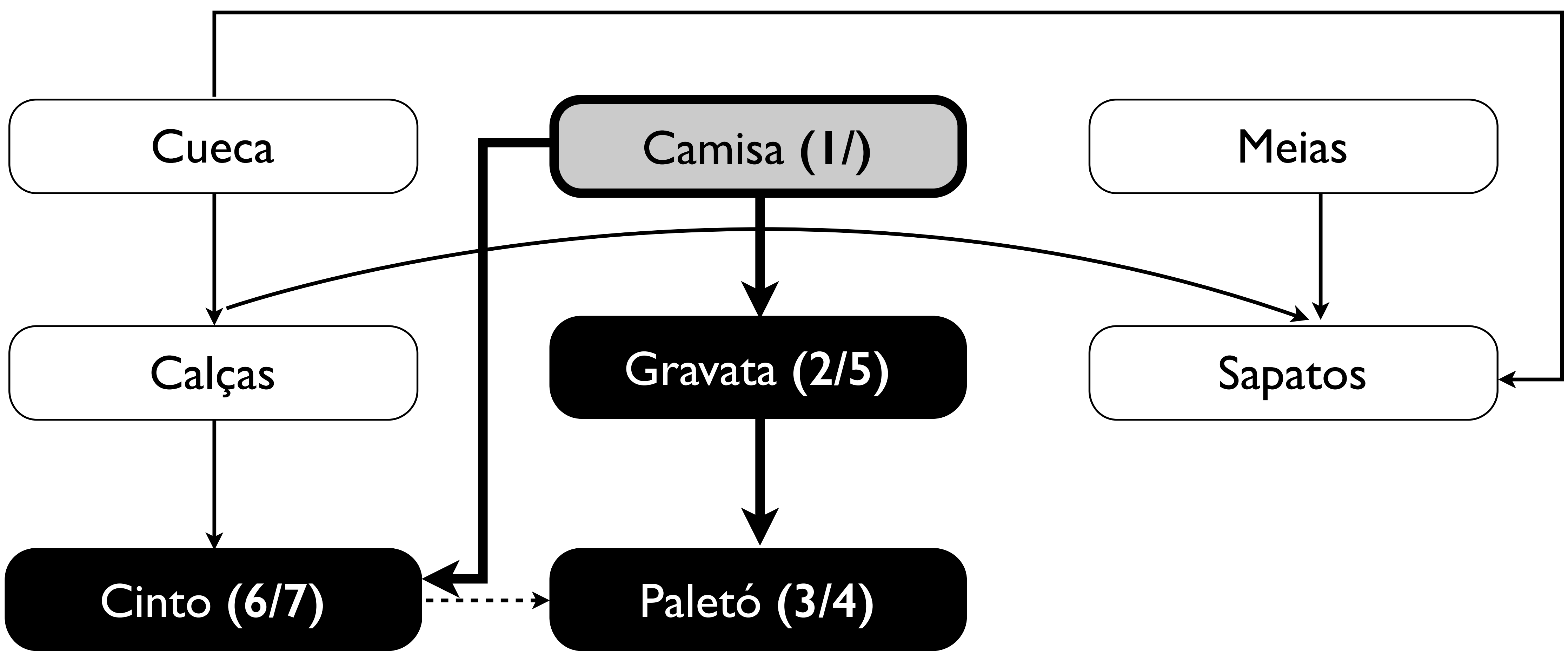
{Gravata, Paletó}

Ordenação Topológica



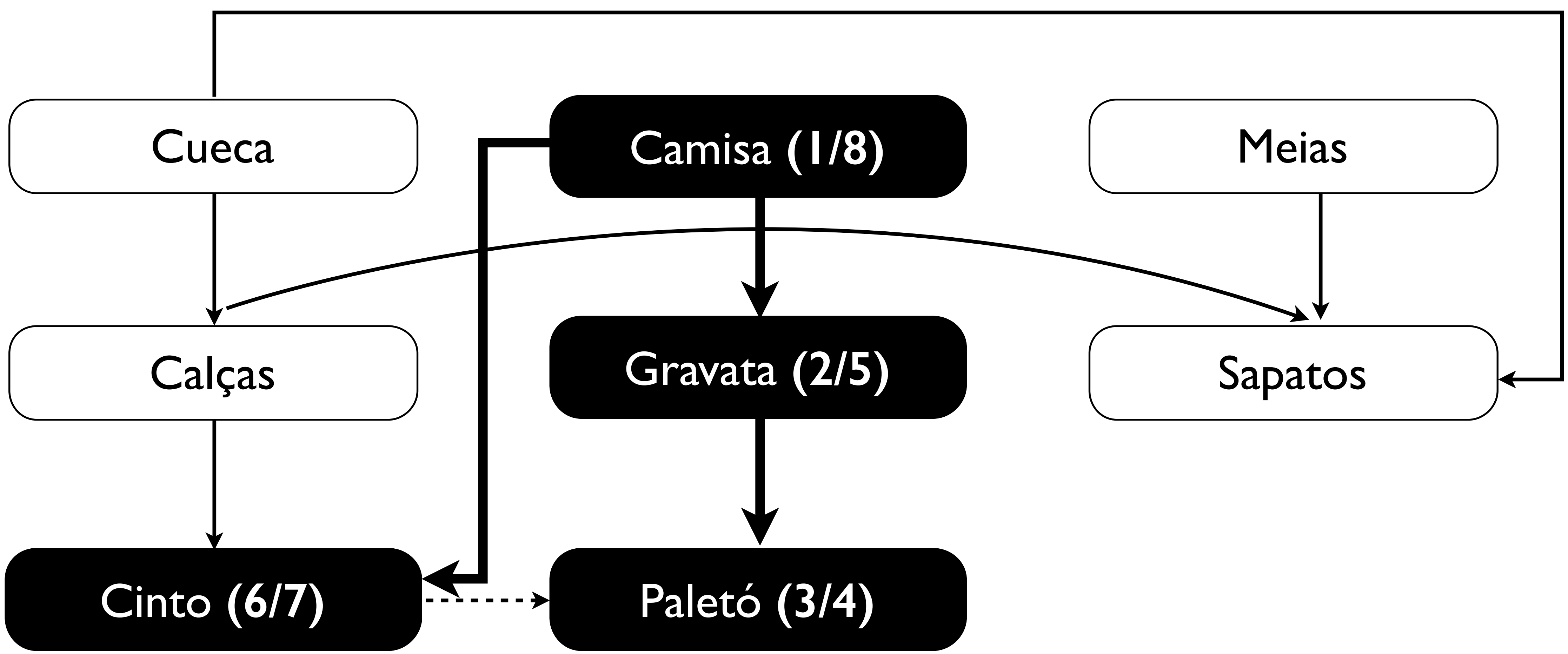
{Gravata, Paletó}

Ordenação Topológica



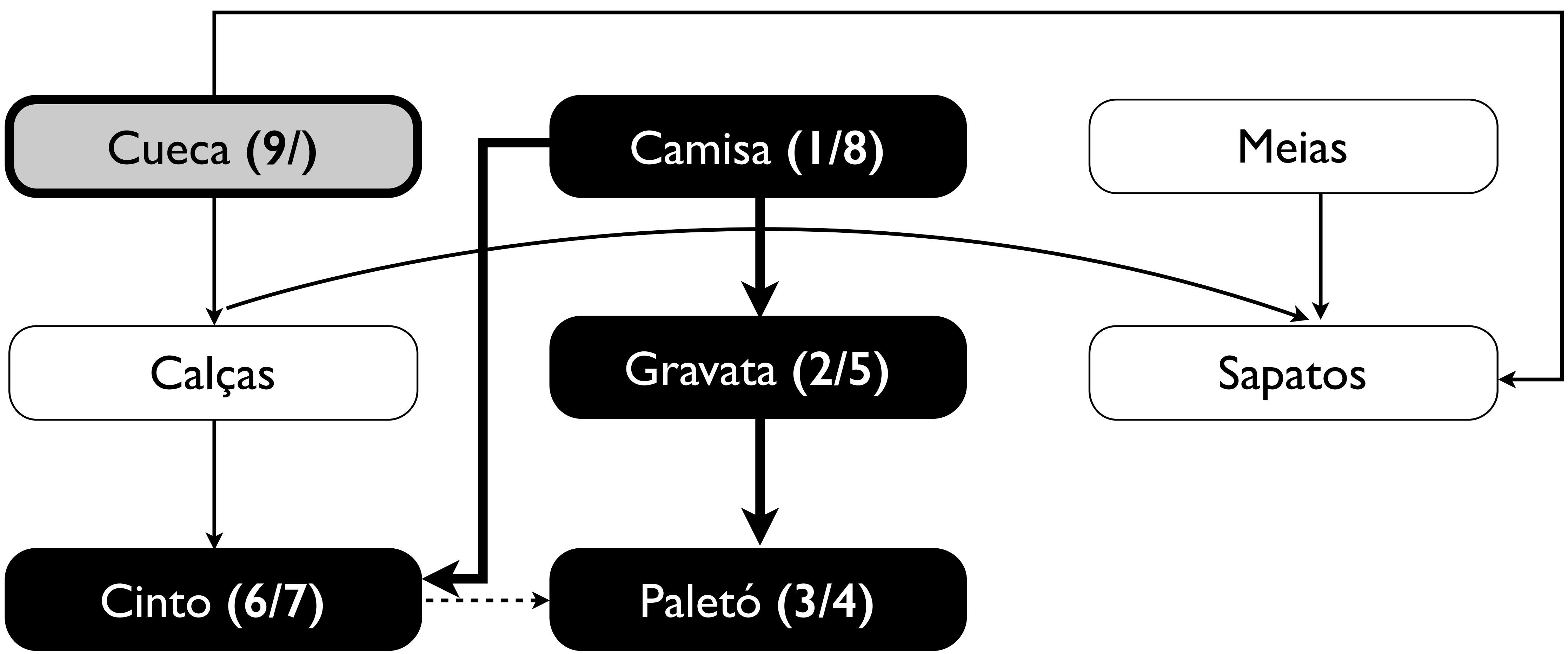
{Cinto, Gravata, Paletó}

Ordenação Topológica



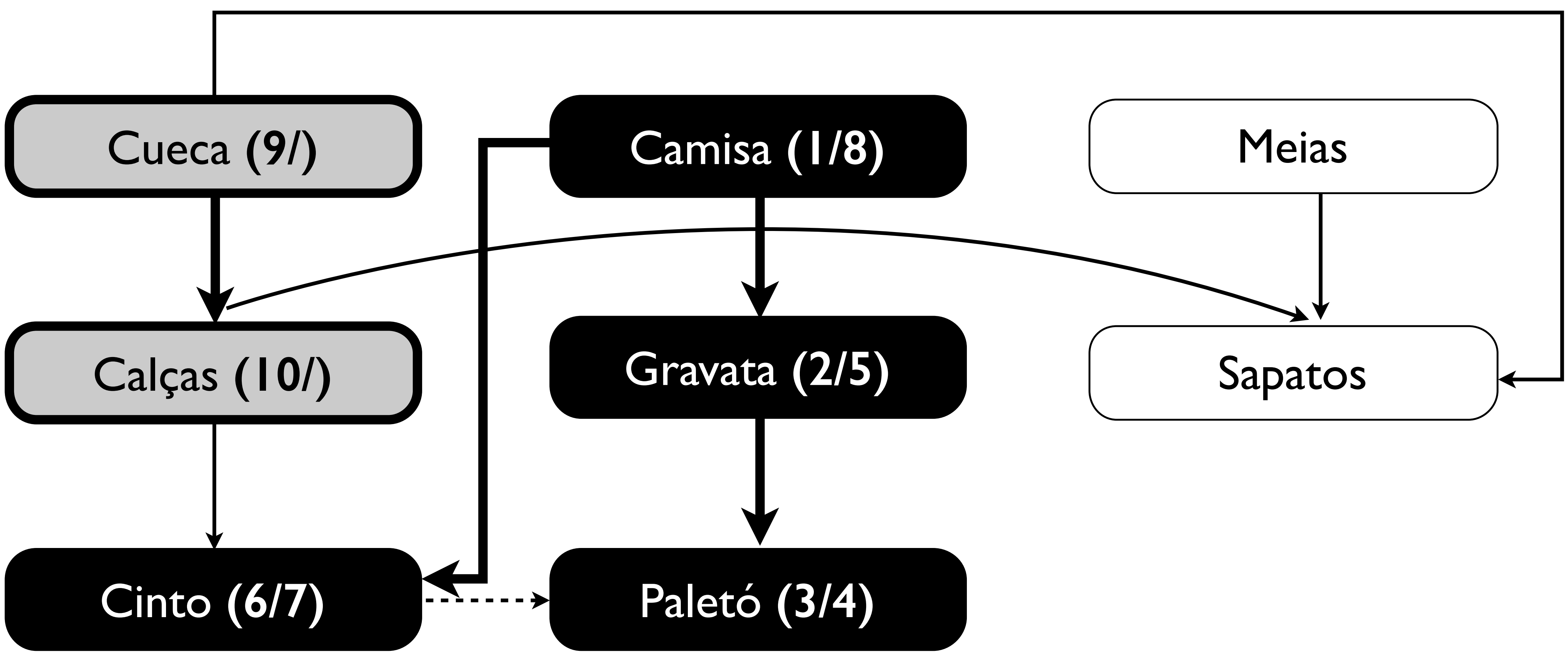
{Camisa, Cinto, Gravata, Paletó}

Ordenação Topológica



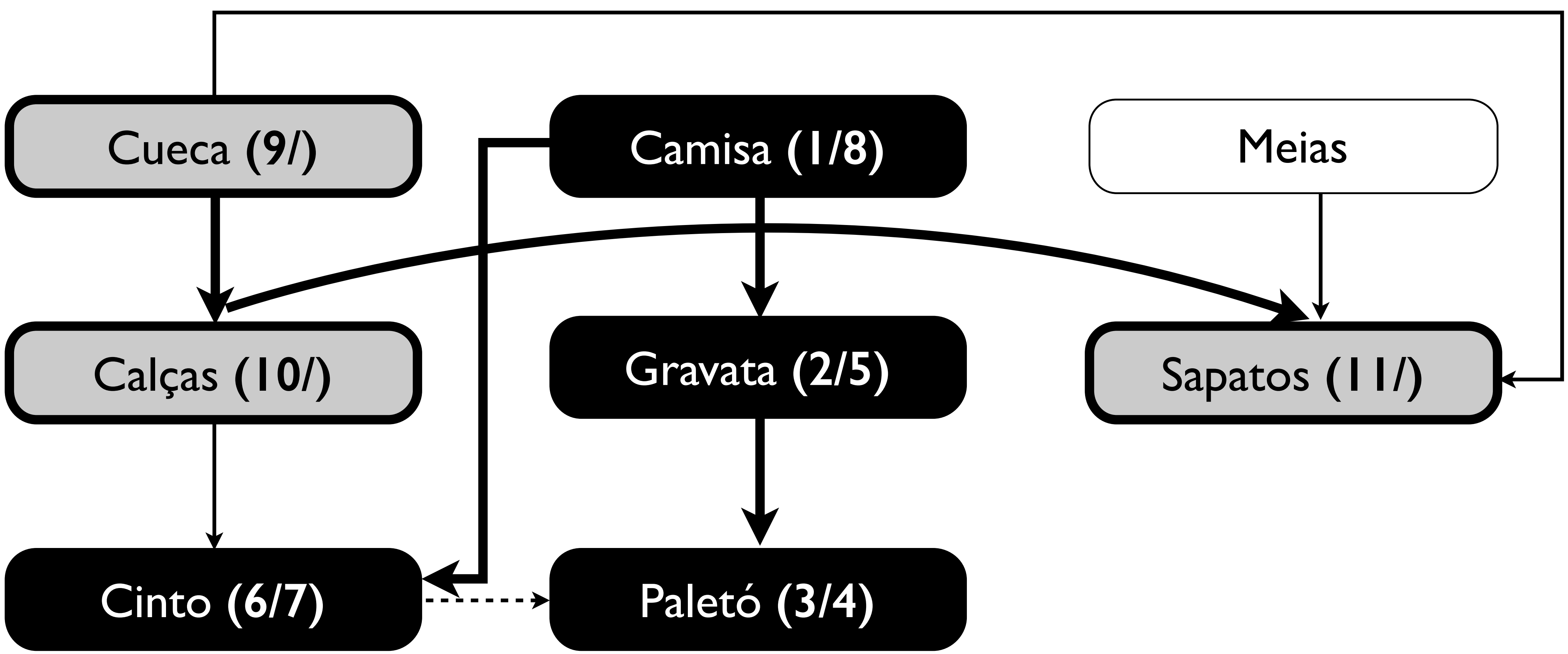
{Camisa, Cinto, Gravata, Paletó}

Ordenação Topológica



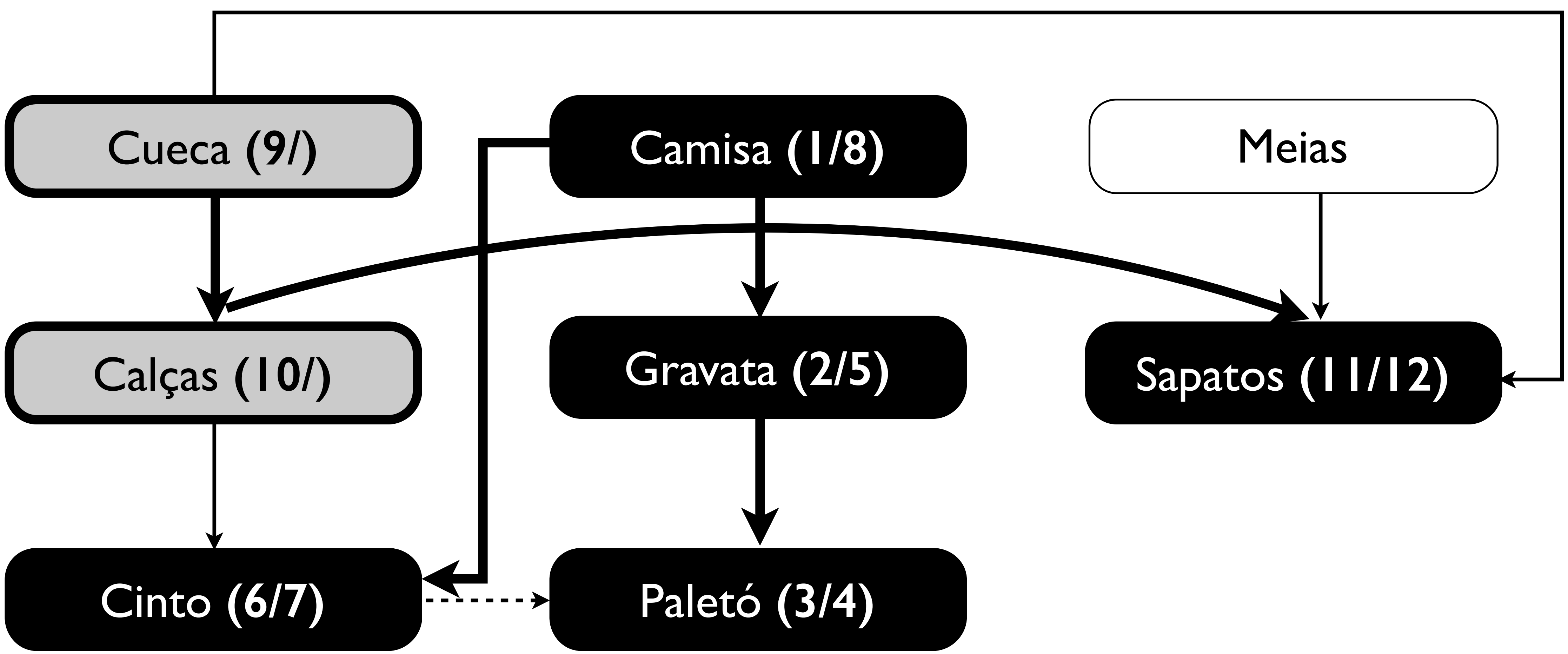
{Camisa, Cinto, Gravata, Paletó}

Ordenação Topológica



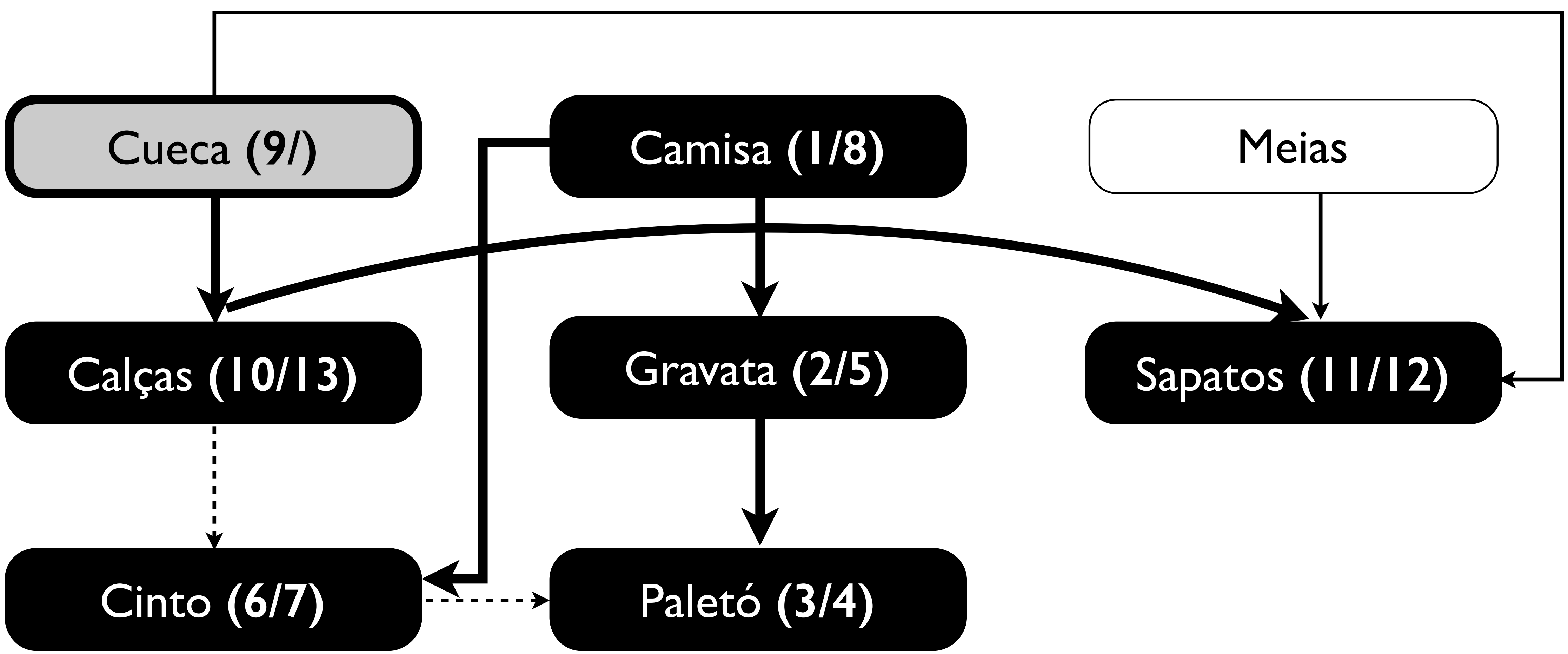
{Camisa, Cinto, Gravata, Paletó}

Ordenação Topológica



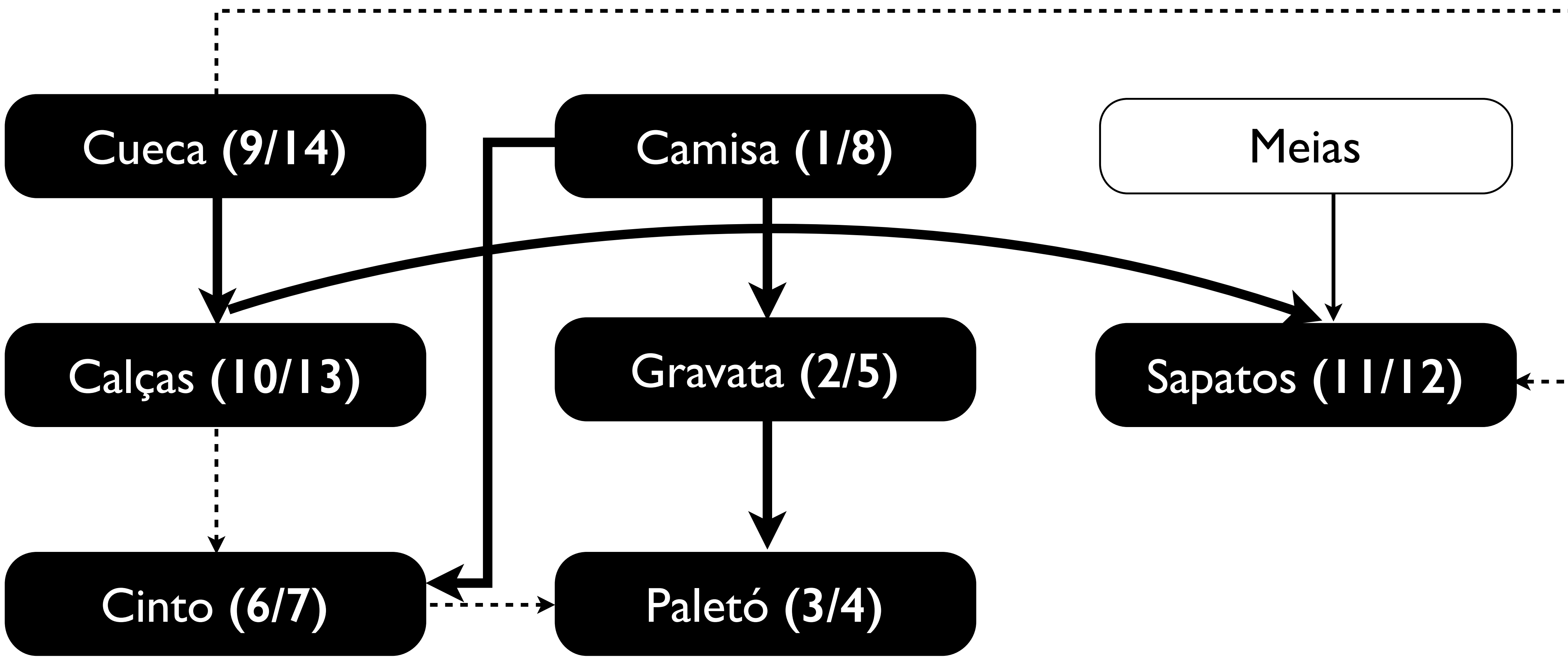
{Sapatos, Camisa, Cinto, Gravata, Paletó}

Ordenação Topológica



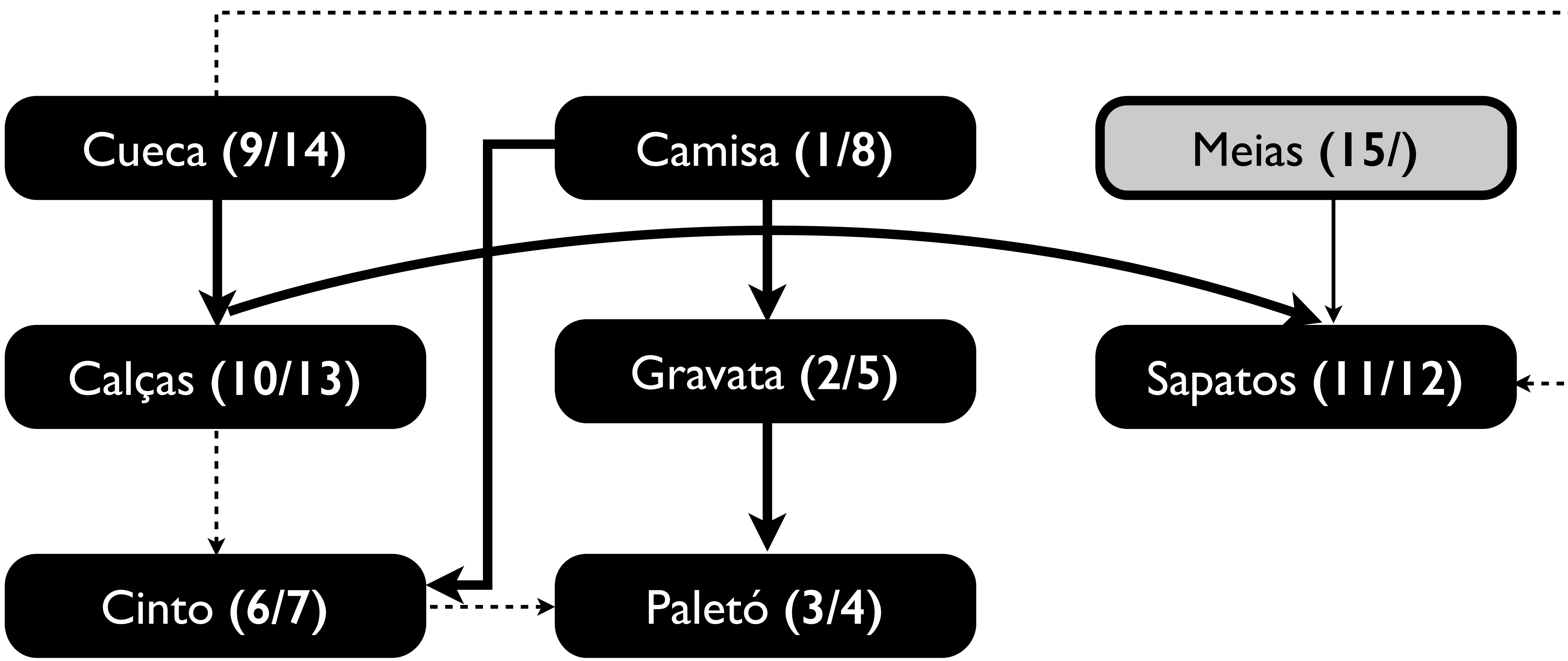
{Calças, Sapatos, Camisa, Cinto, Gravata, Paletó}

Ordenação Topológica



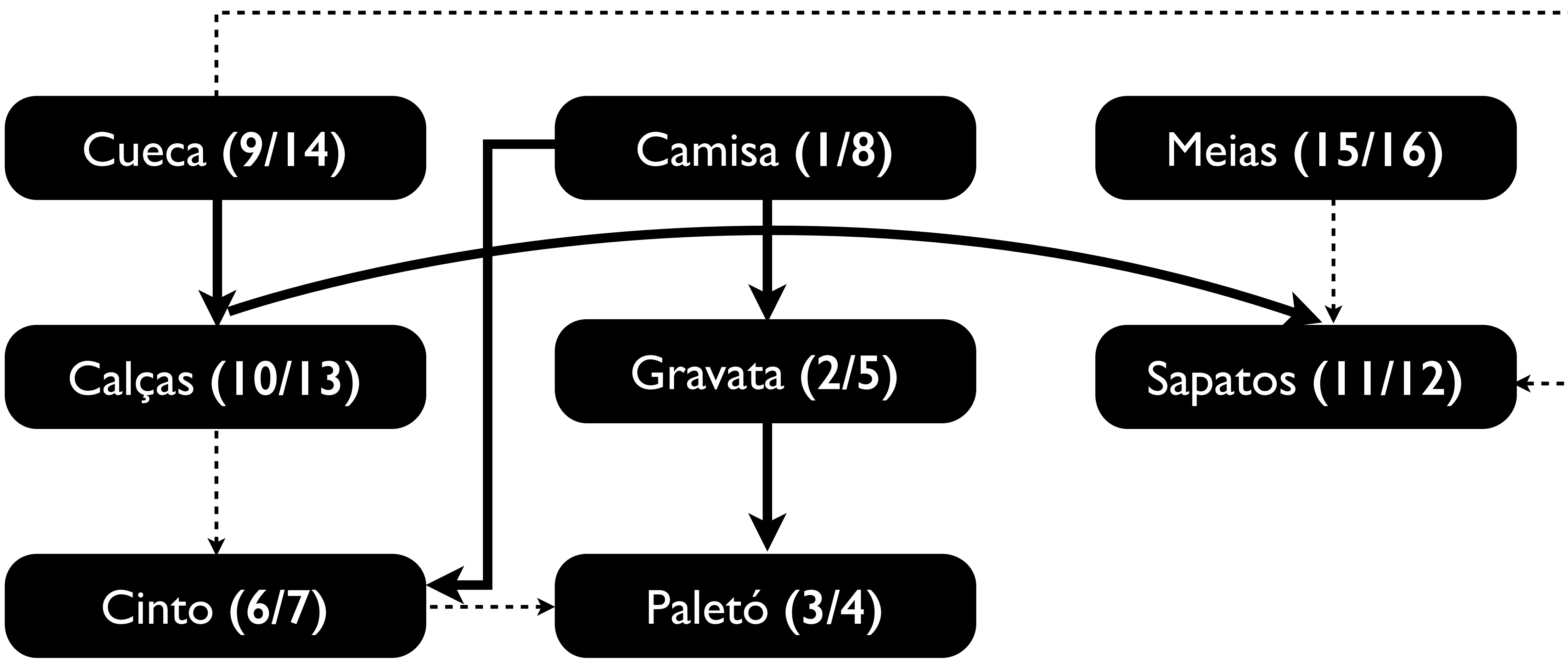
{Cueca, Calças, Sapatos, Camisa, Cinto, Gravata, Paletó}

Ordenação Topológica



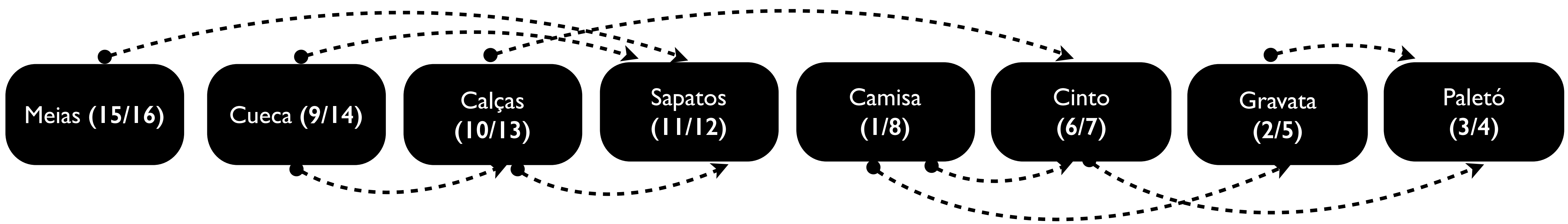
{Cueca, Calças, Sapatos, Camisa, Cinto, Gravata, Paletó}

Ordenação Topológica



{Meias, Cueca, Calças, Sapatos, Camisa, Cinto, Gravata, Paletó}

Ordenação Topológica



{Meias, Cueca, Calças, Sapatos, Camisa, Cinto, Gravata, Paletó}

Componentes Fortemente Conectados

Aplicação 2

Componentes Conectados

- A decomposição de um grafo orientado em seus componentes fortemente conectados
- Por que? Porque muitos algoritmos podem fazer a decomposição e depois serem executados separadamente em cada componente

Componentes Conectados

- O que é um componente fortemente conectado em um grafo G = (V, A)?
  - É um conjunto maximal de vértices **C** em V, tal que para todo par *u* e *v* em **C**, temos ao mesmo tempo um caminho de *u* a *v* e *v* a *u*
  - Em outras palavras, *u* e *v* são mutuamente acessíveis

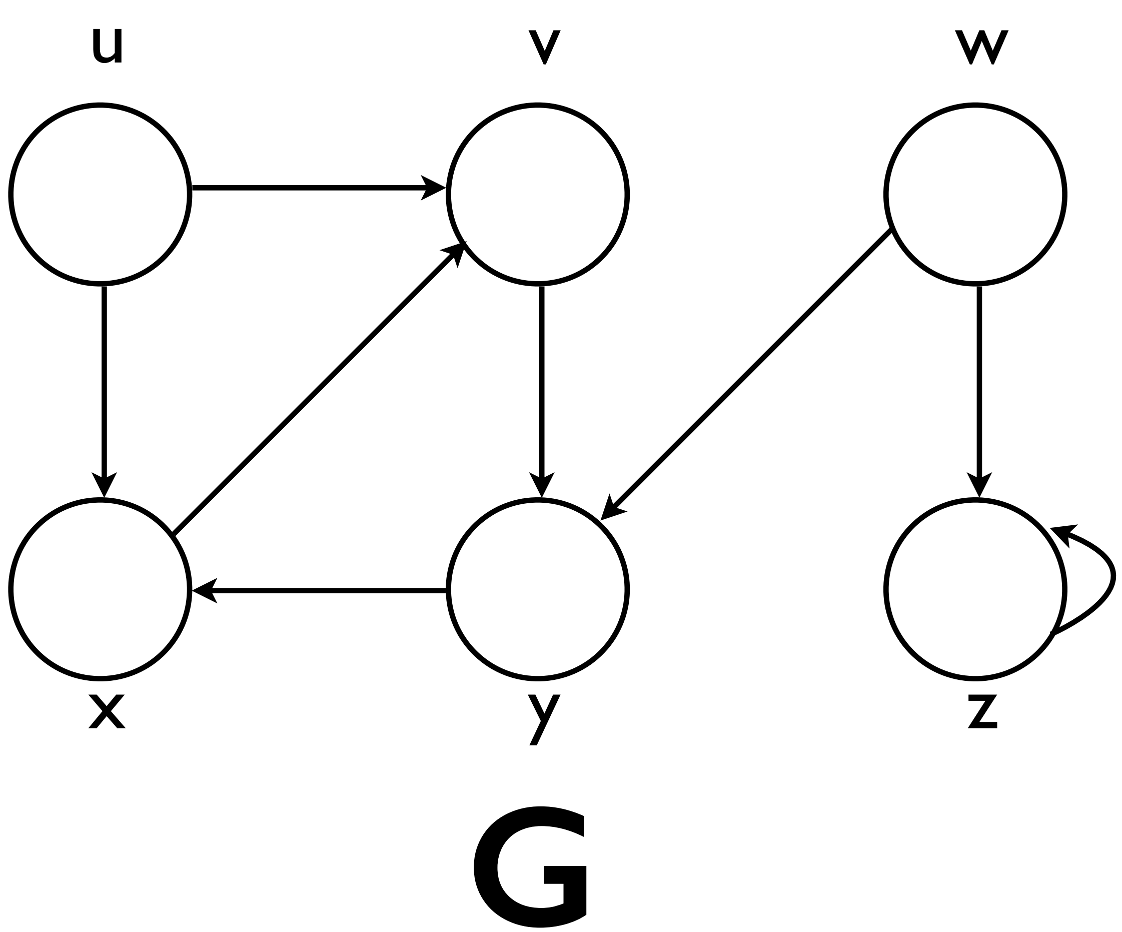
Componentes Conectados

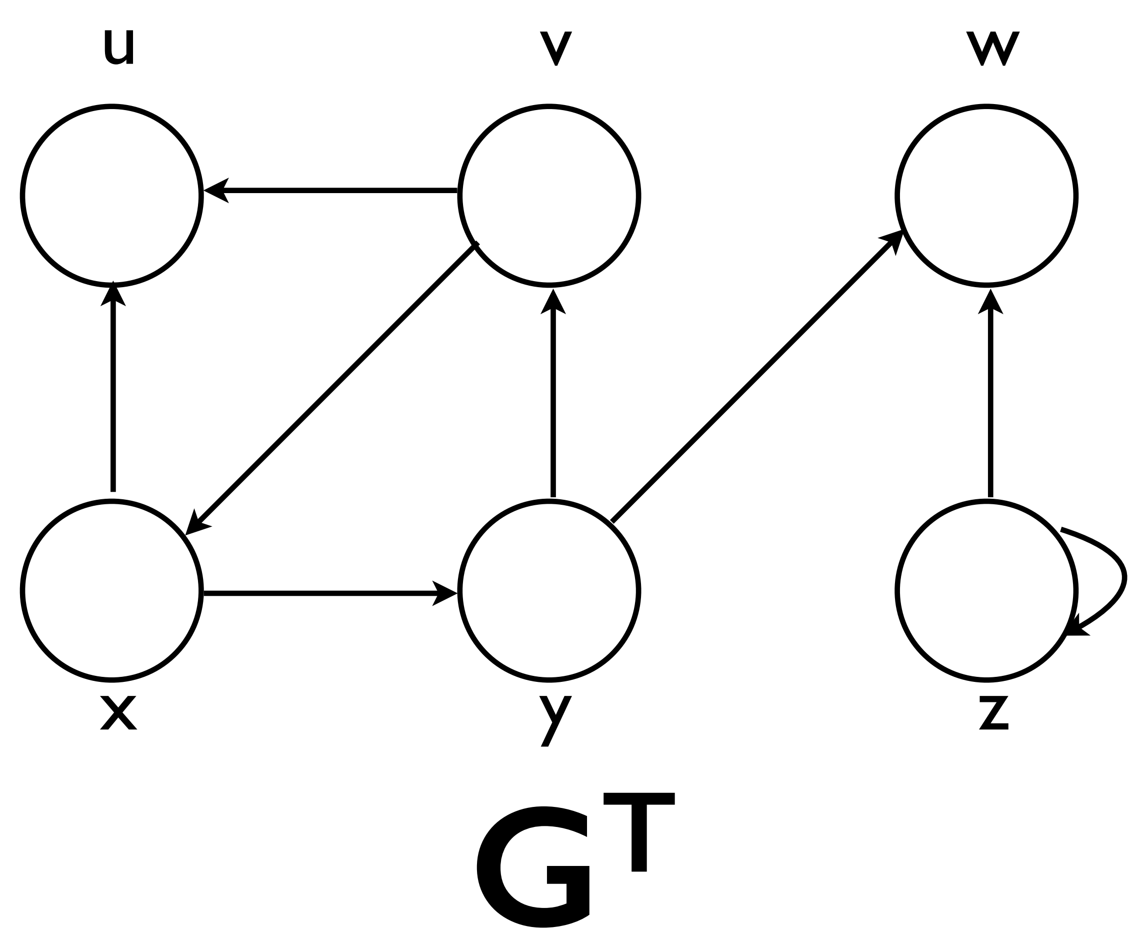
1. **CPTFortementeConectado**(G)
2. BuscaProf(G)
3. Calcular G^T
4. BuscaProf(G^T), mas no *laço* principal da busca em profundidade, considerar os vértices em ordem decrescente de tempo de término f_u
5. Dar saída aos vértices de cada árvore resultante em (3) como um CPT fortemente conectado

Componentes Conectados

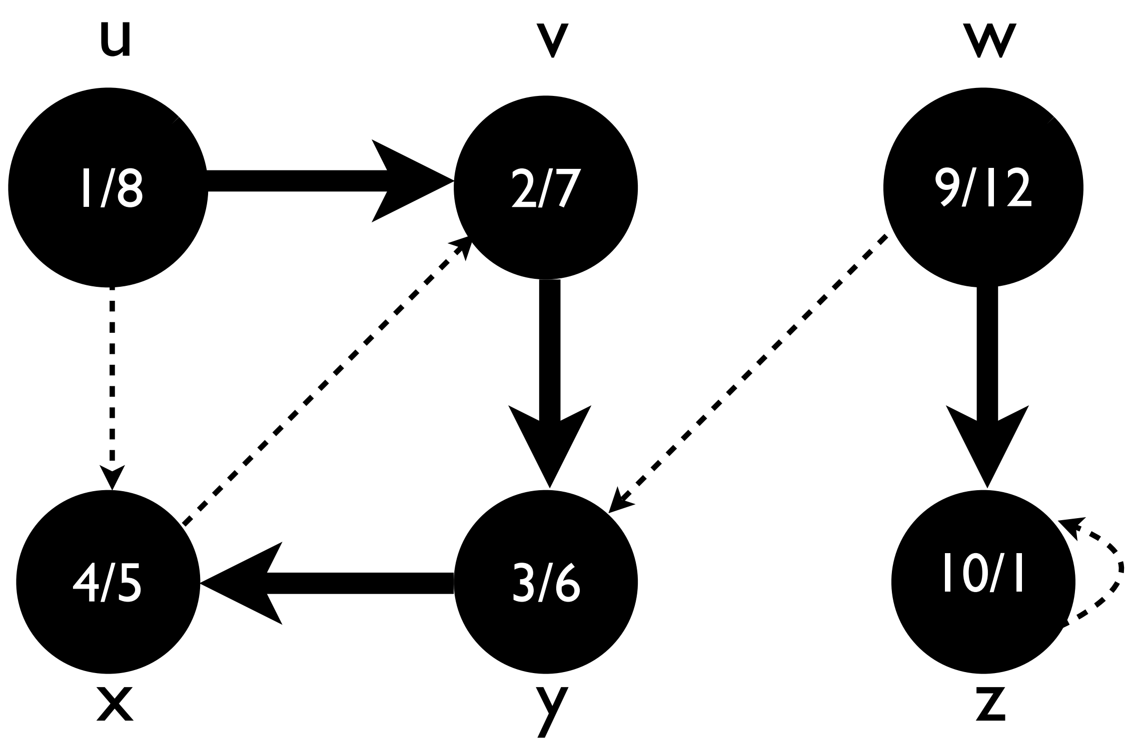
1. **CPTFortementeConectado**(G)
2. BuscaProf(G)
3. Calcular G^T
4. BuscaProf(G^T), mas no *laço* principal da busca em profundidade, considerar os vértices em ordem decrescente de tempo de término f_u
5. Dar saída aos vértices de cada árvore resultante em (3) como um CPT fortemente conectado

Componentes Conectados

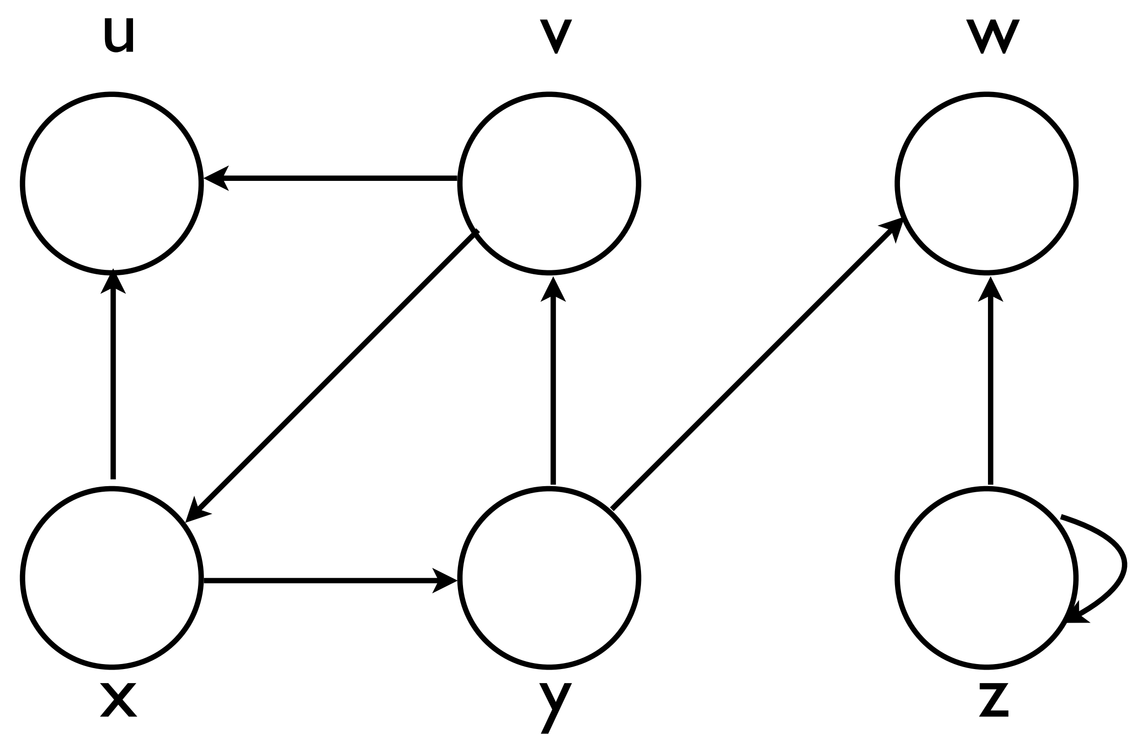




Componentes Conectados

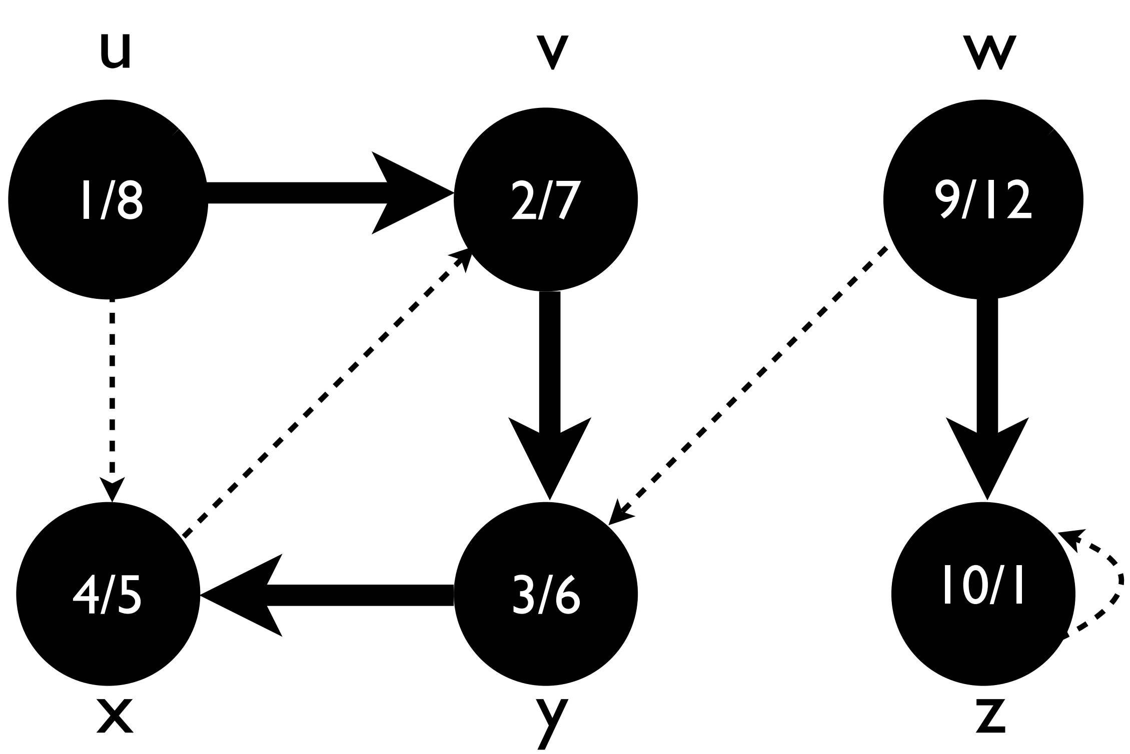


BuscaProf(G)

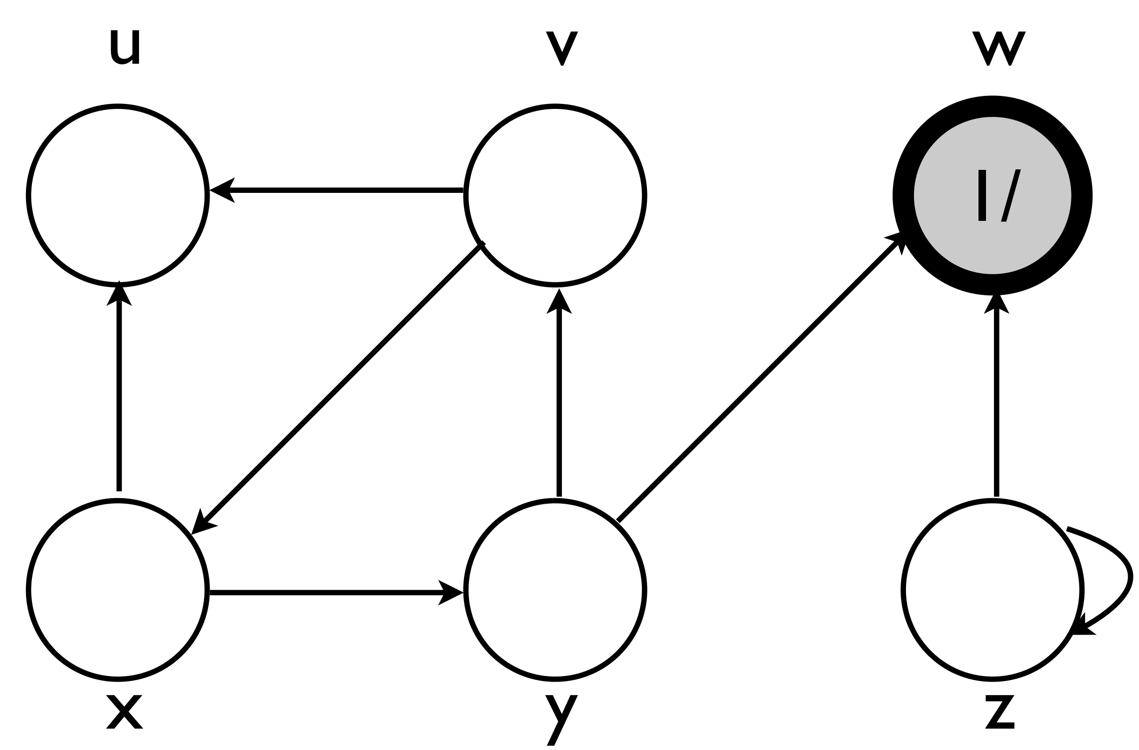


BuscaProf(G^T)

Componentes Conectados

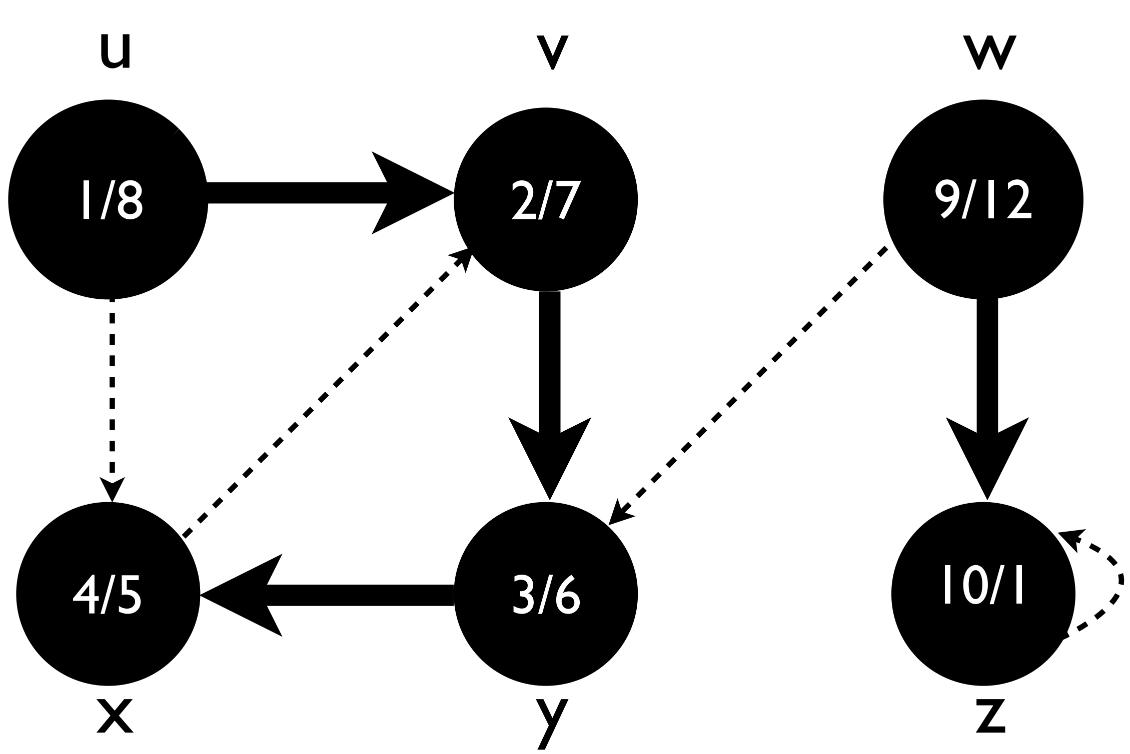


BuscaProf(G)

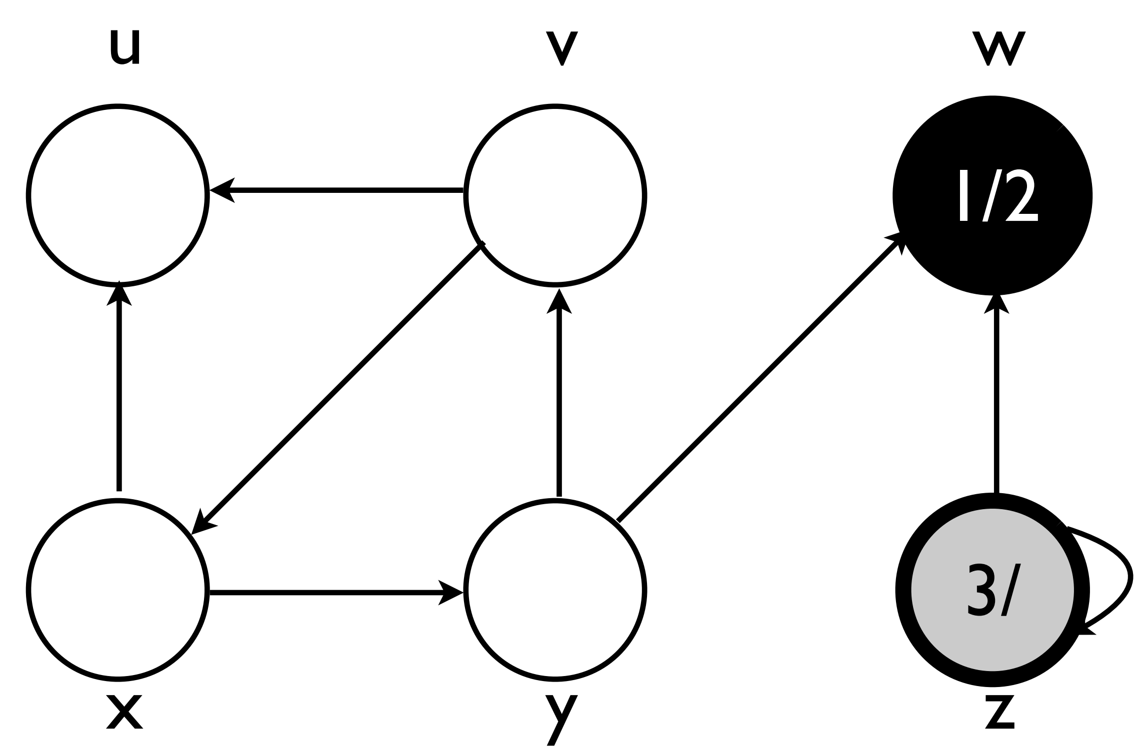


BuscaProf(G^T)

Componentes Conectados

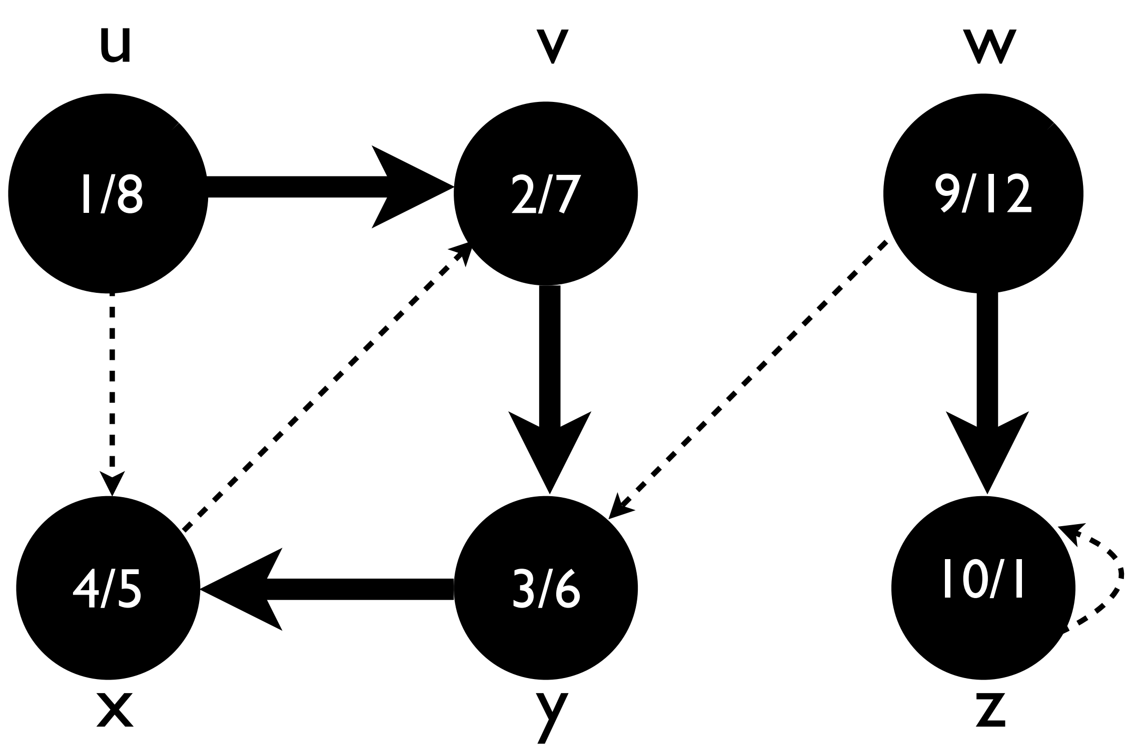


BuscaProf(G)

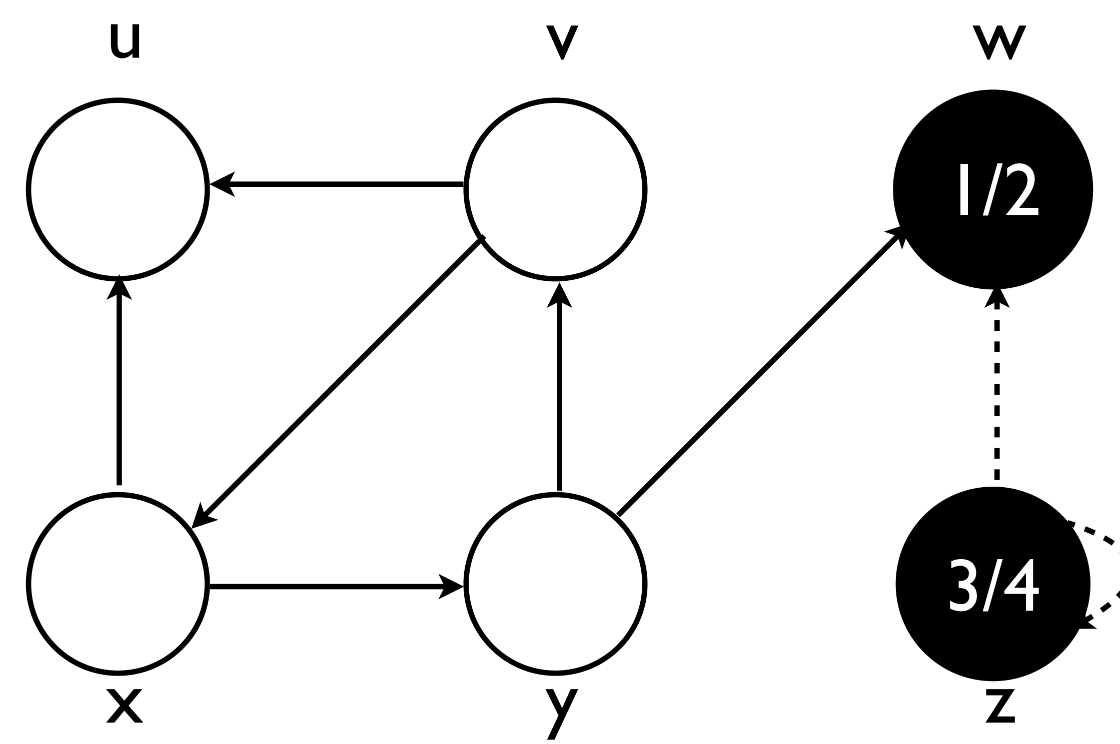


BuscaProf(G^T)

Componentes Conectados

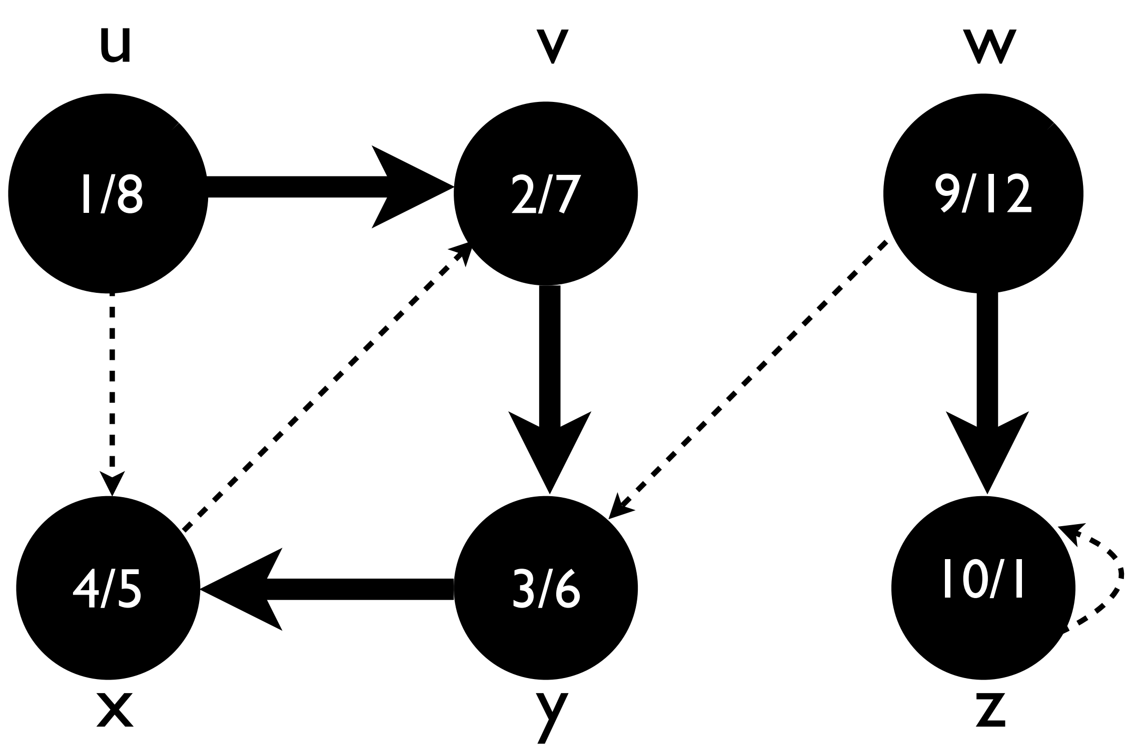


BuscaProf(G)

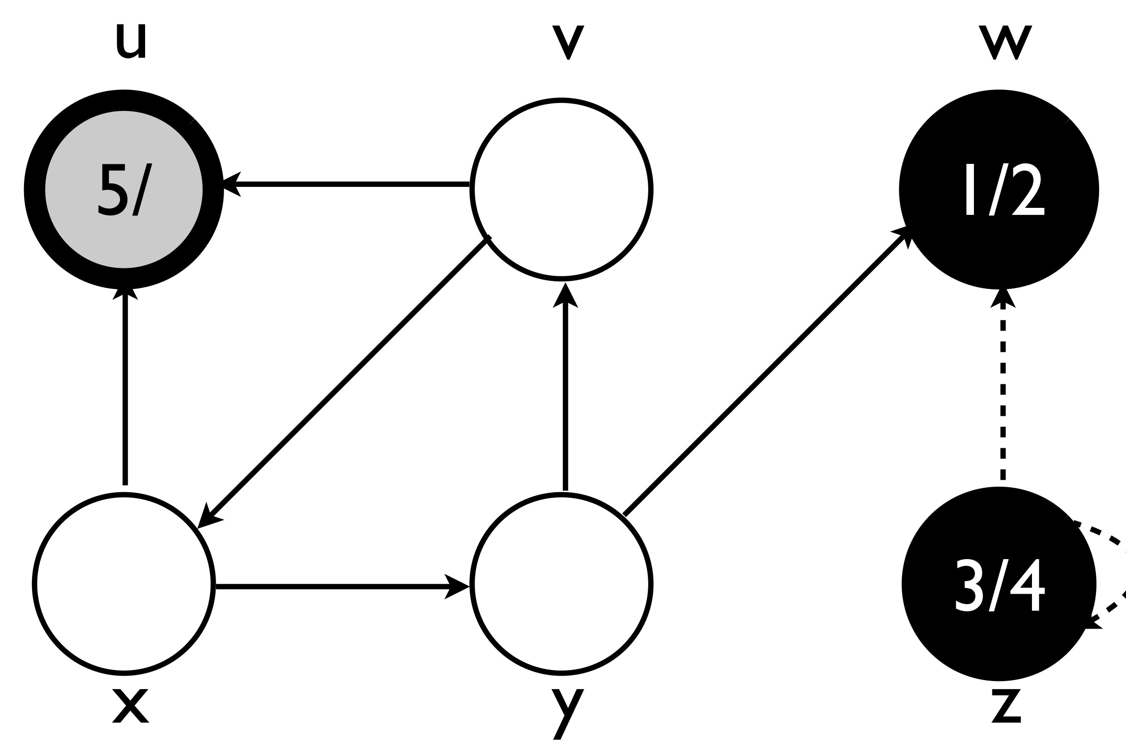


BuscaProf(G^T)

Componentes Conectados

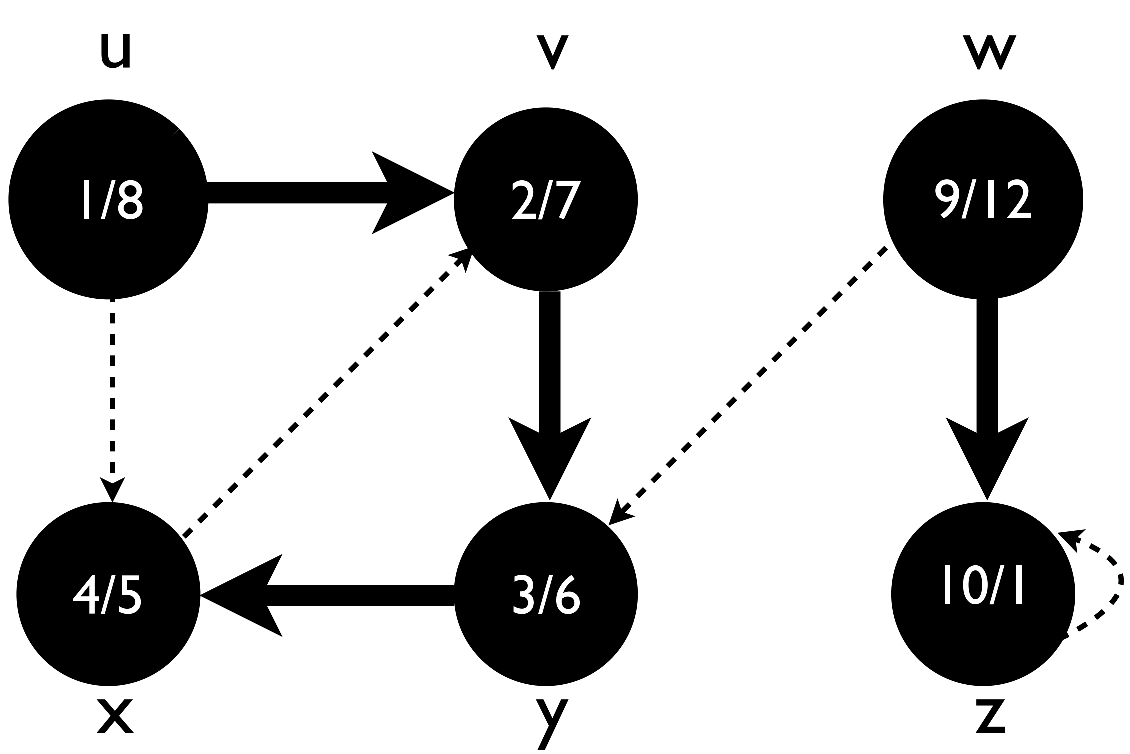


BuscaProf(G)

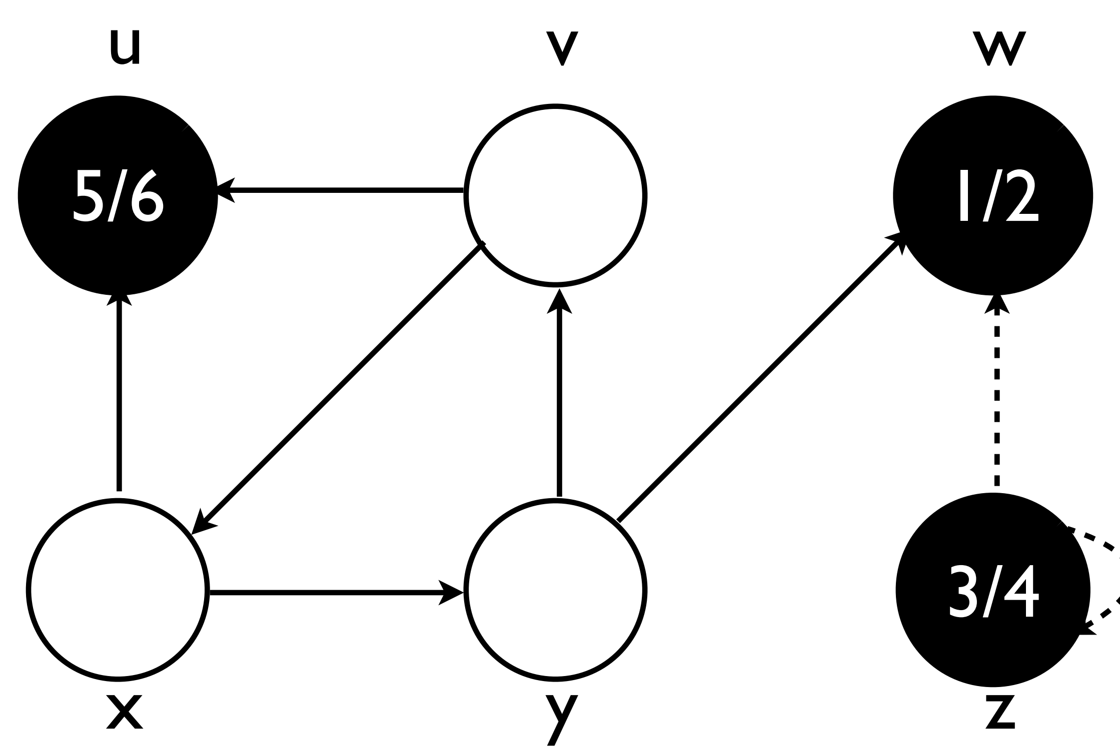


BuscaProf(G^T)

Componentes Conectados

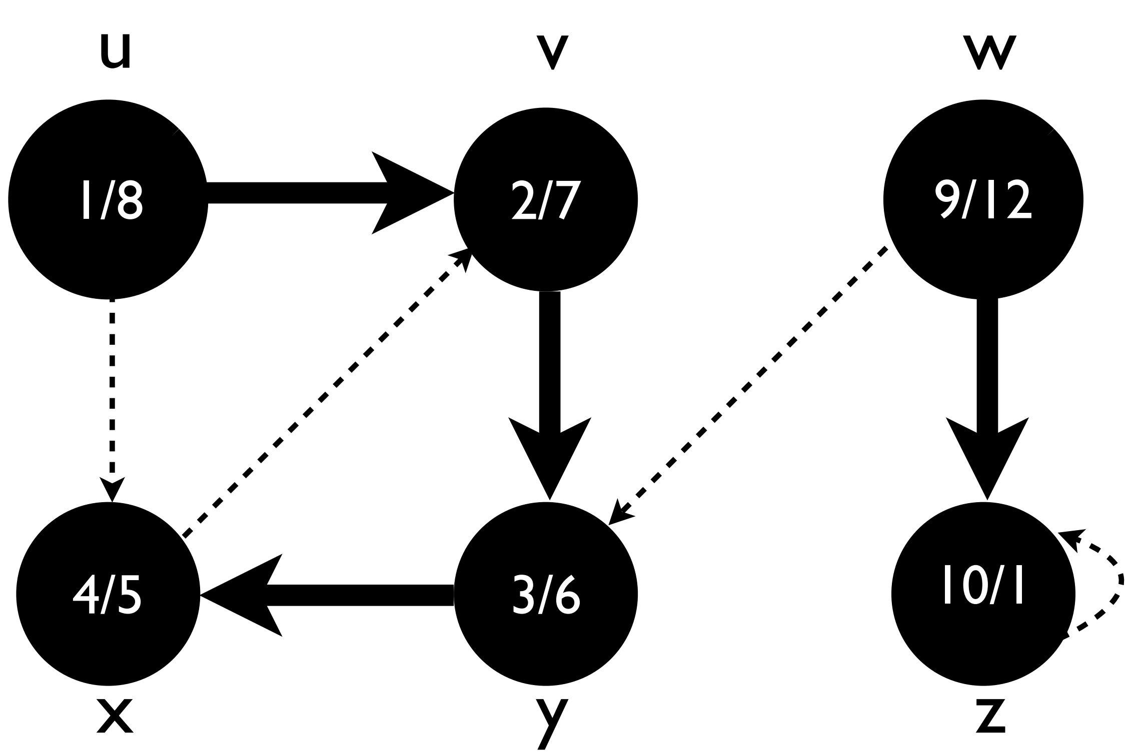


BuscaProf(G)

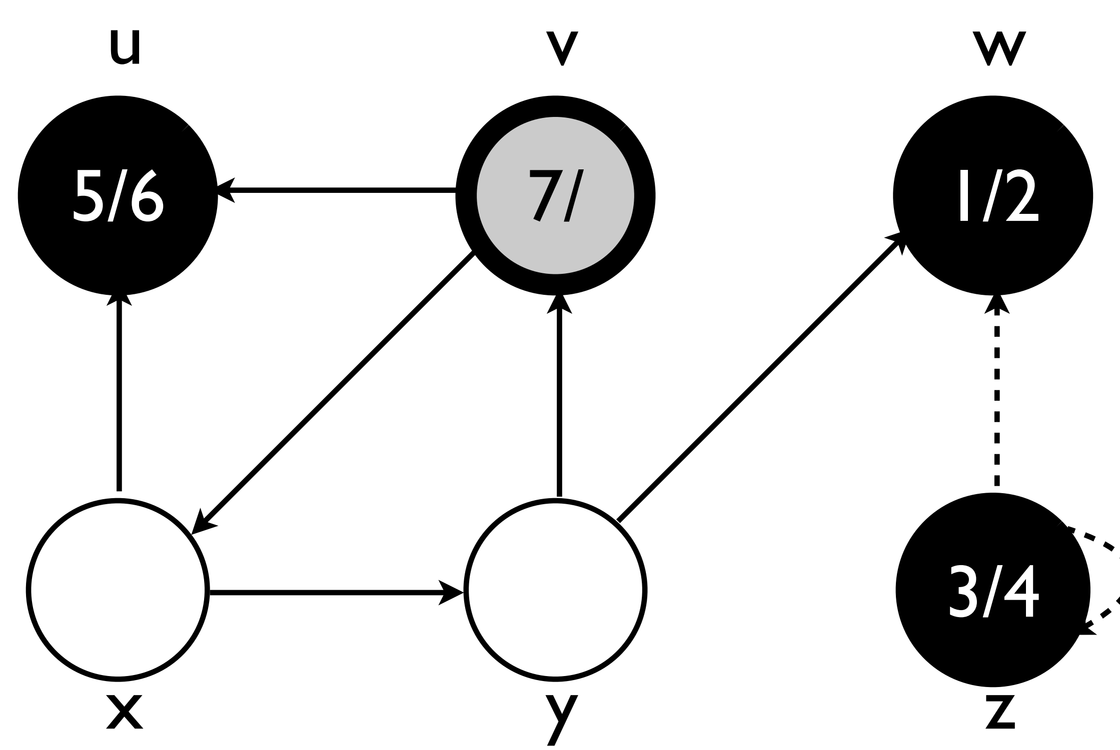


BuscaProf(G^T)

Componentes Conectados

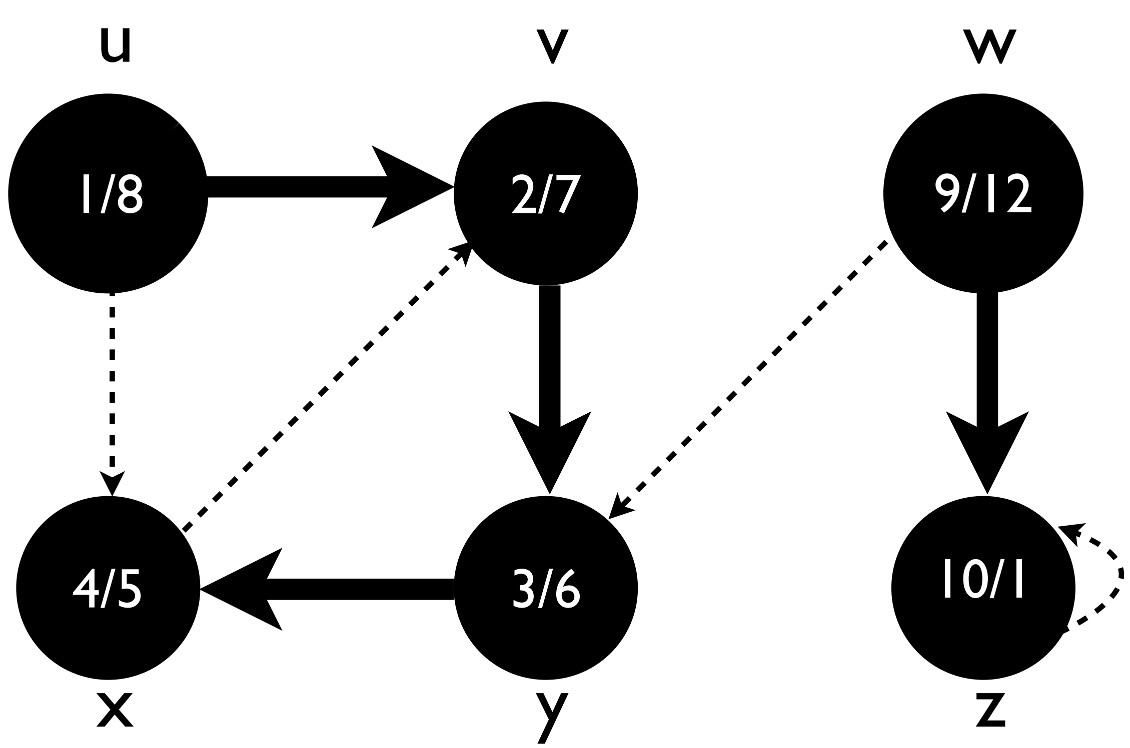


BuscaProf(G)

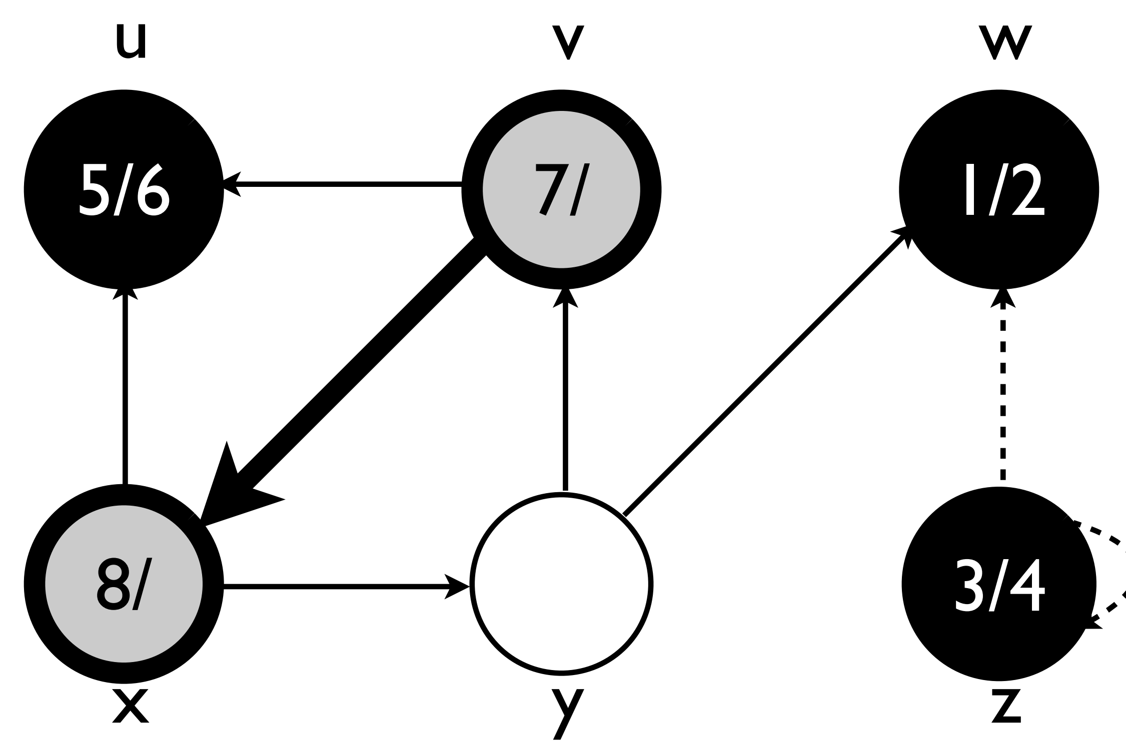


BuscaProf(G^T)

Componentes Conectados

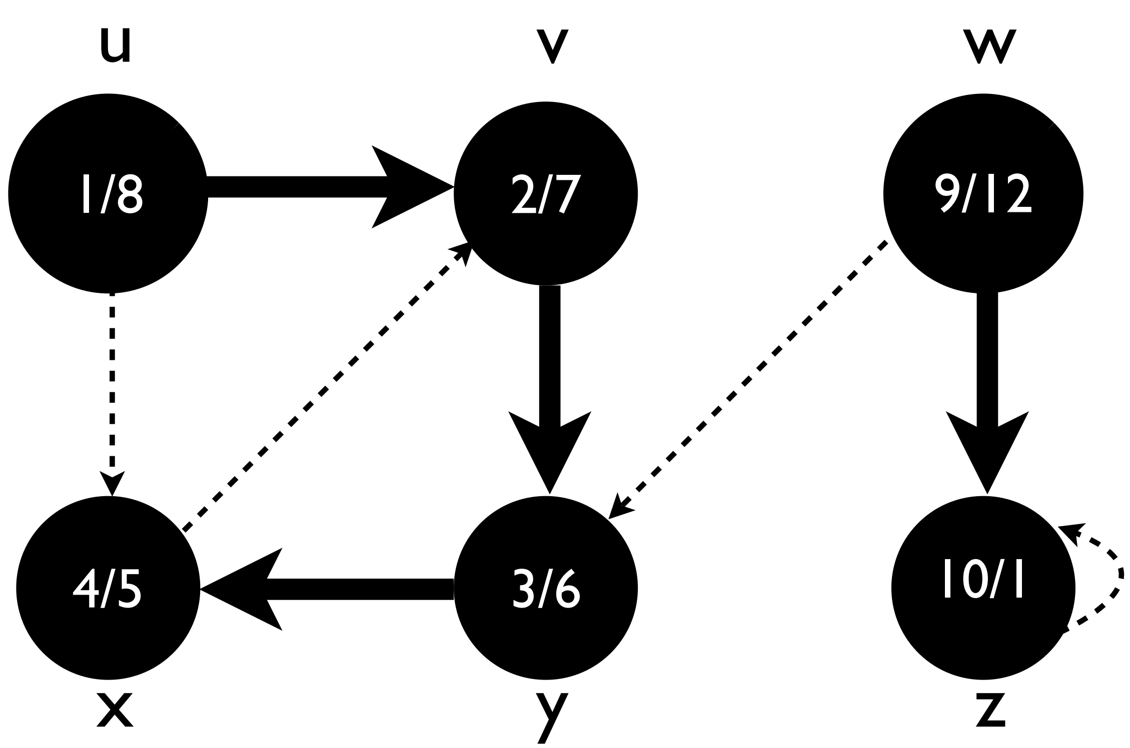


BuscaProf(G)

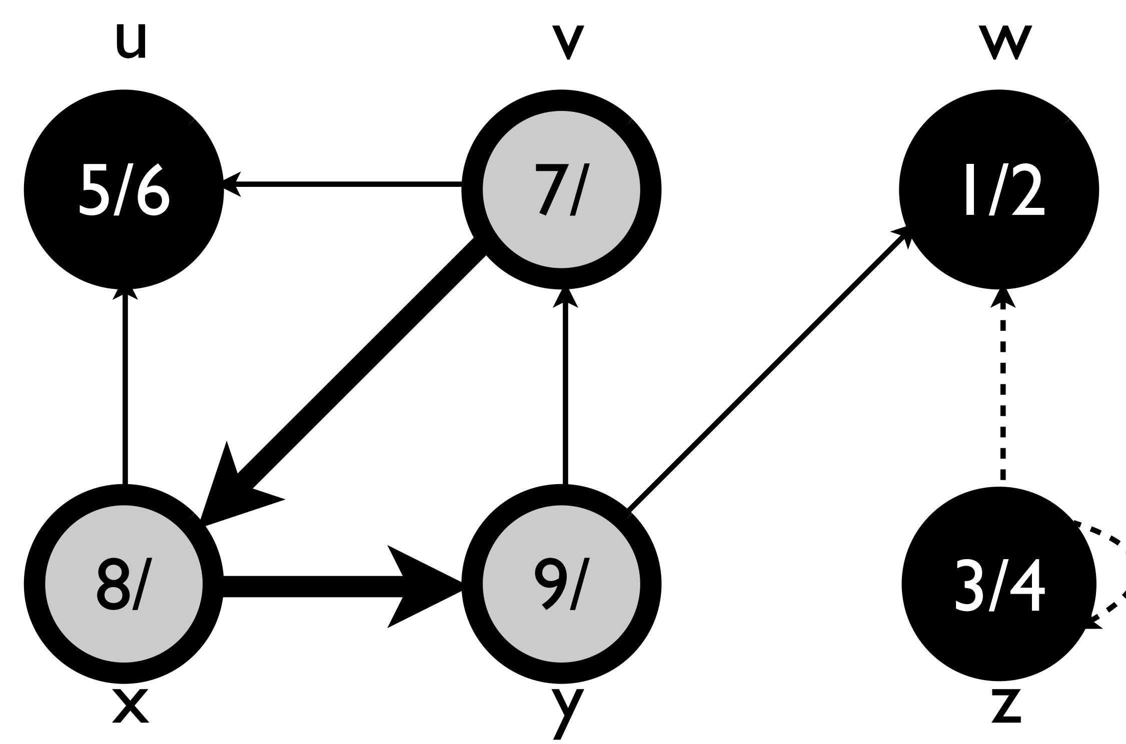


BuscaProf(G^T)

Componentes Conectados

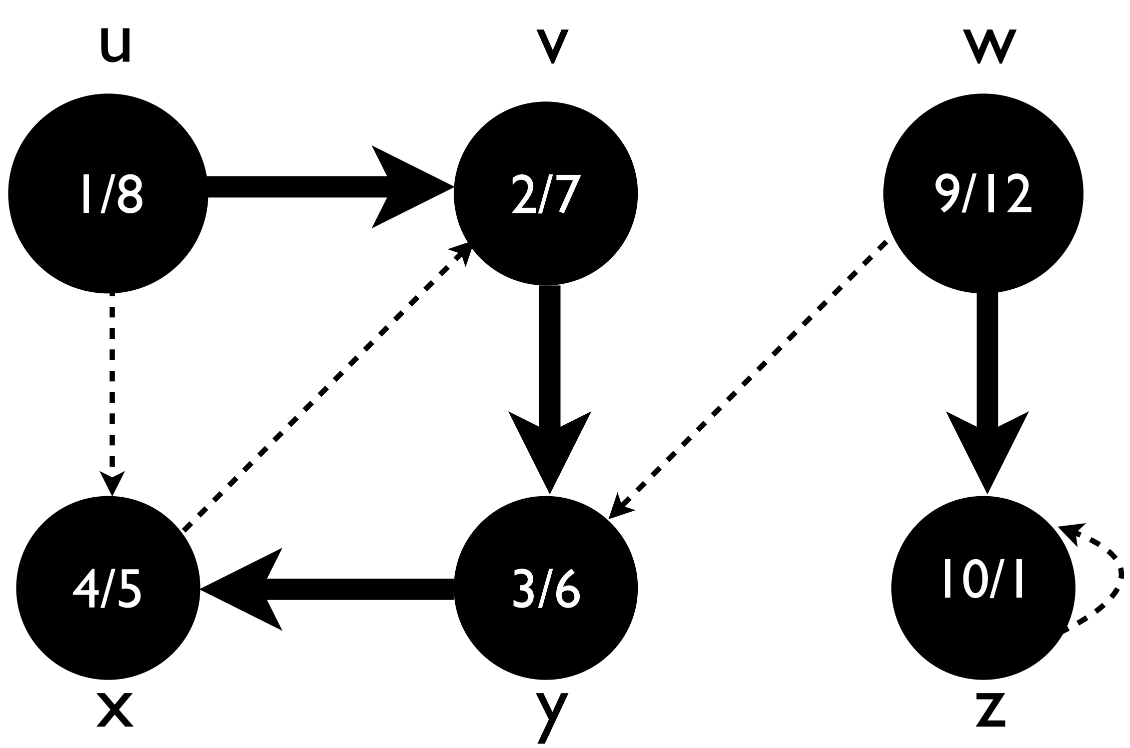


BuscaProf(G)

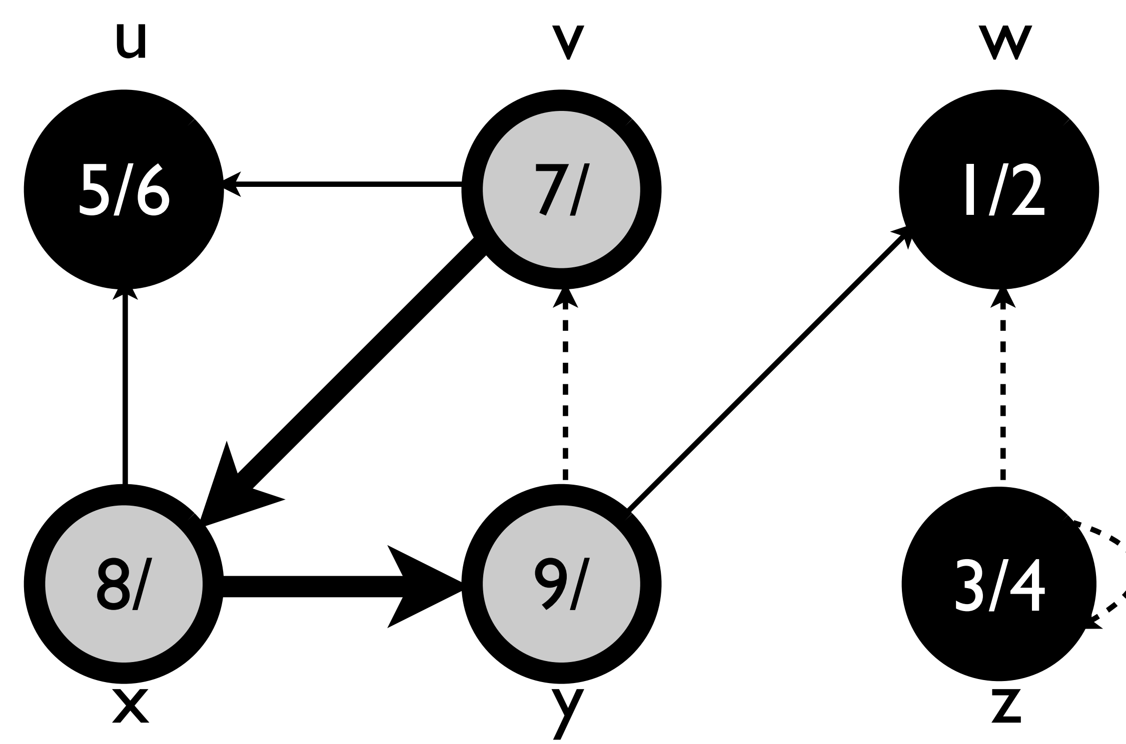


BuscaProf(G^T)

Componentes Conectados

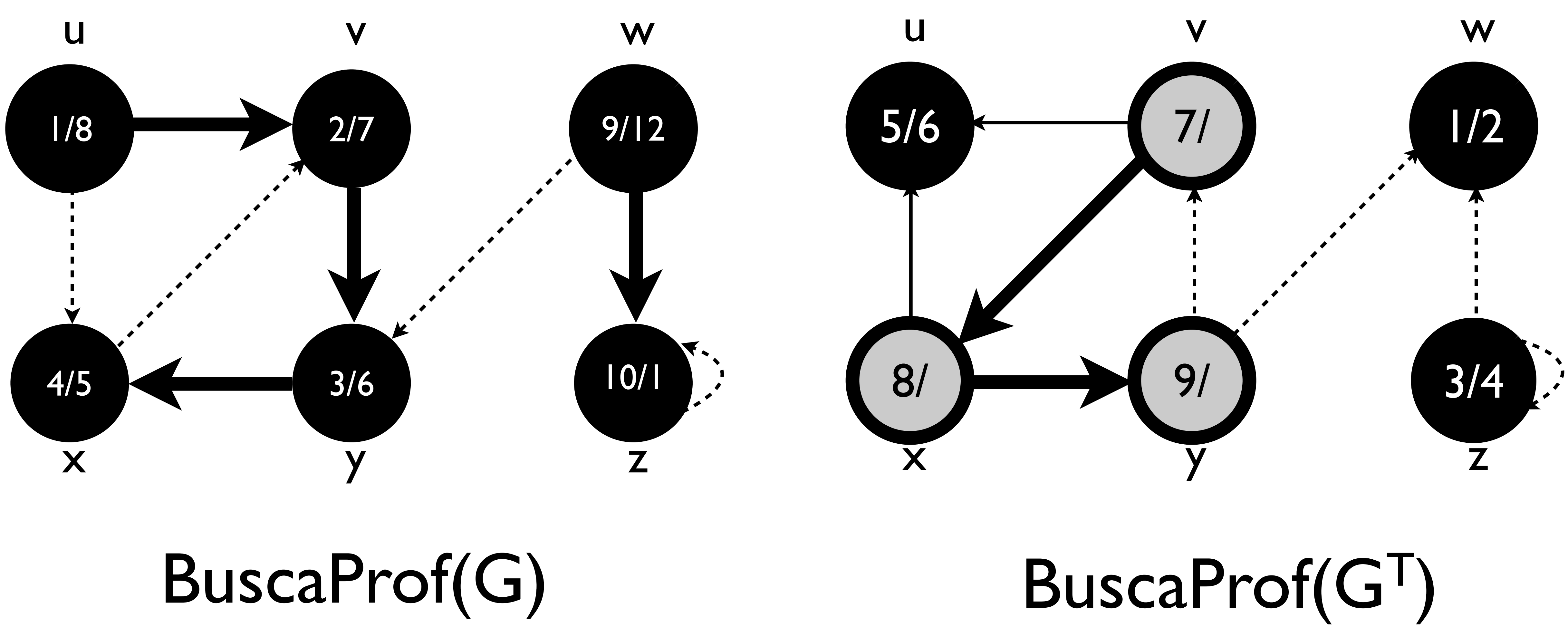


BuscaProf(G)

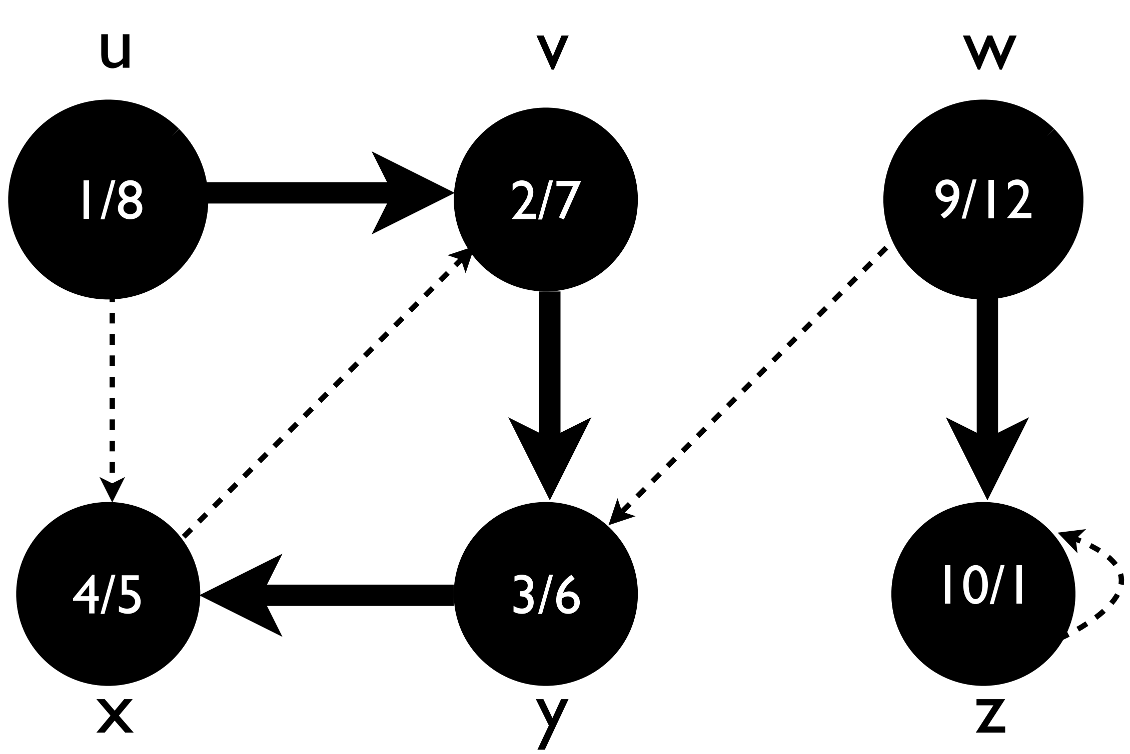


BuscaProf(G^T)

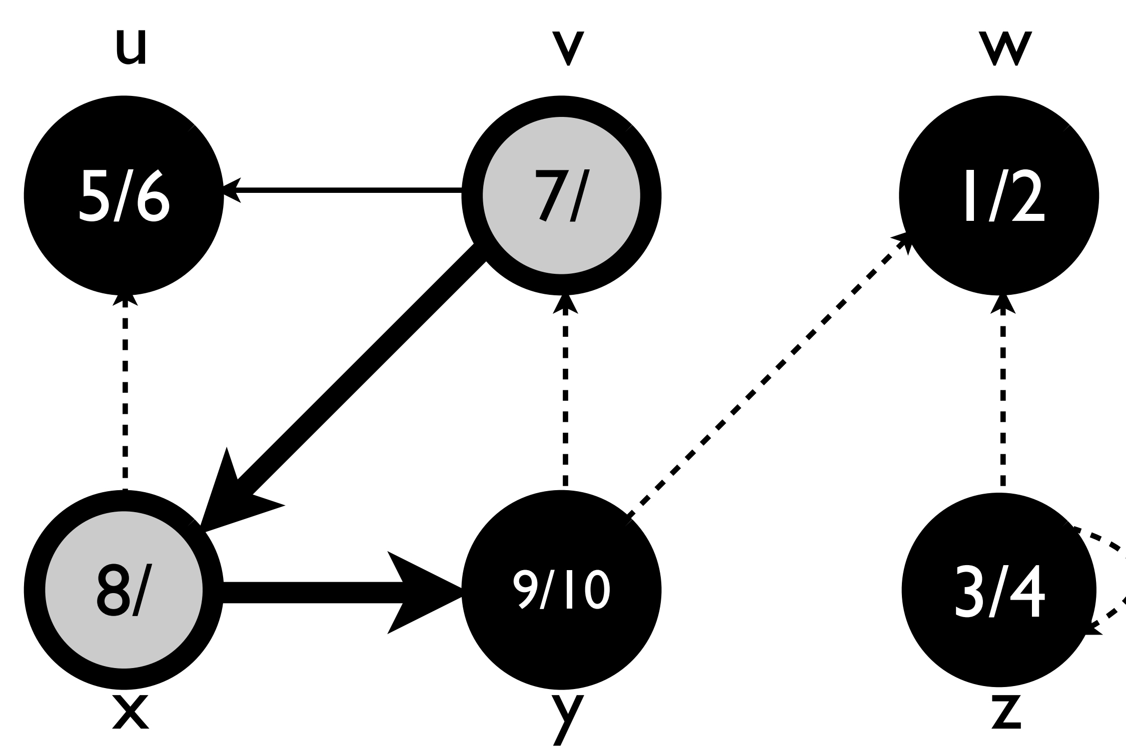
Componentes Conectados



Componentes Conectados

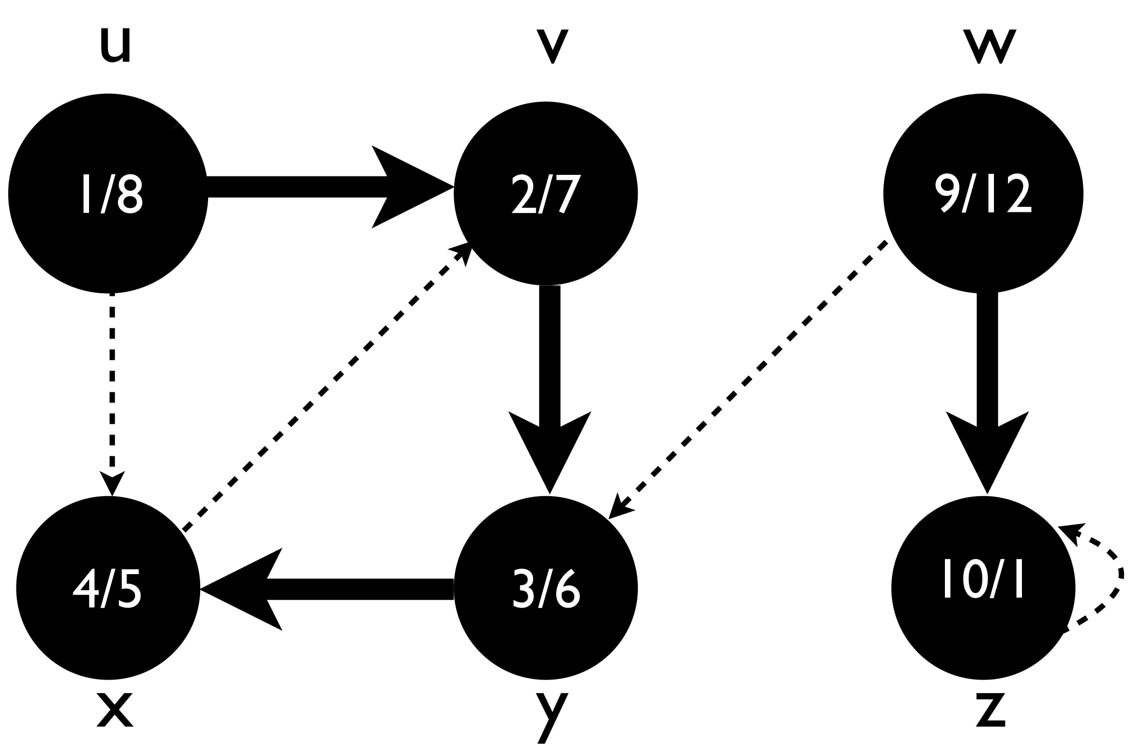


BuscaProf(G)

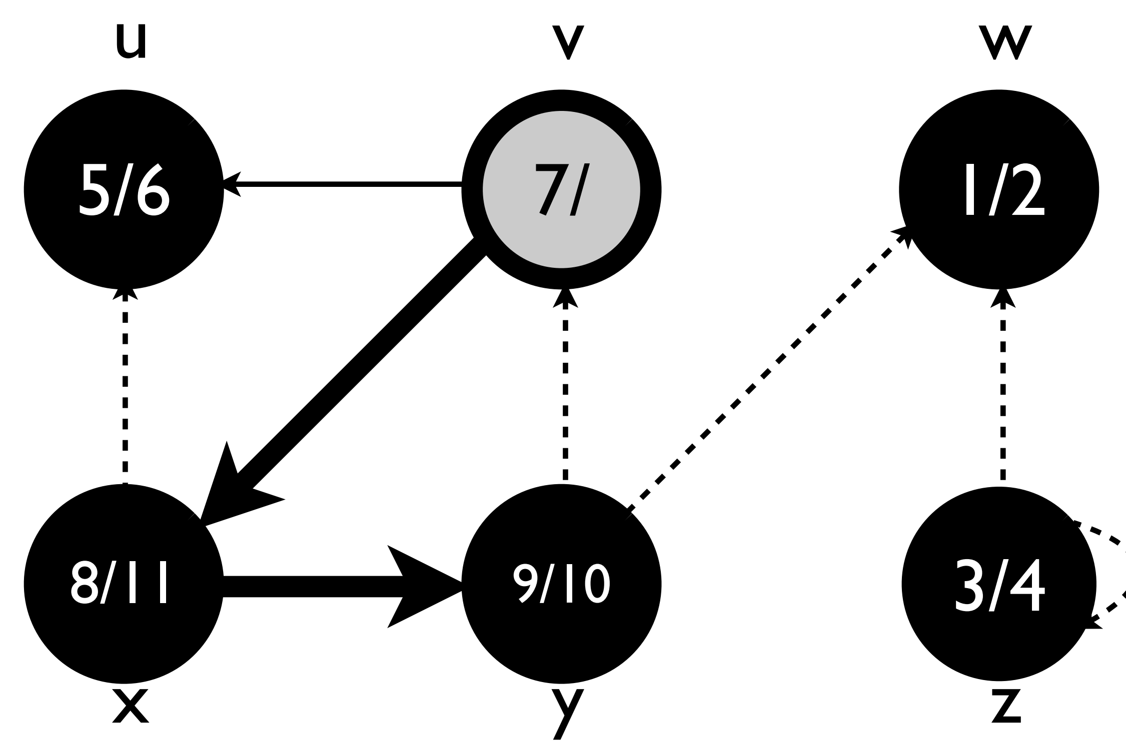


BuscaProf(G^T)

Componentes Conectados

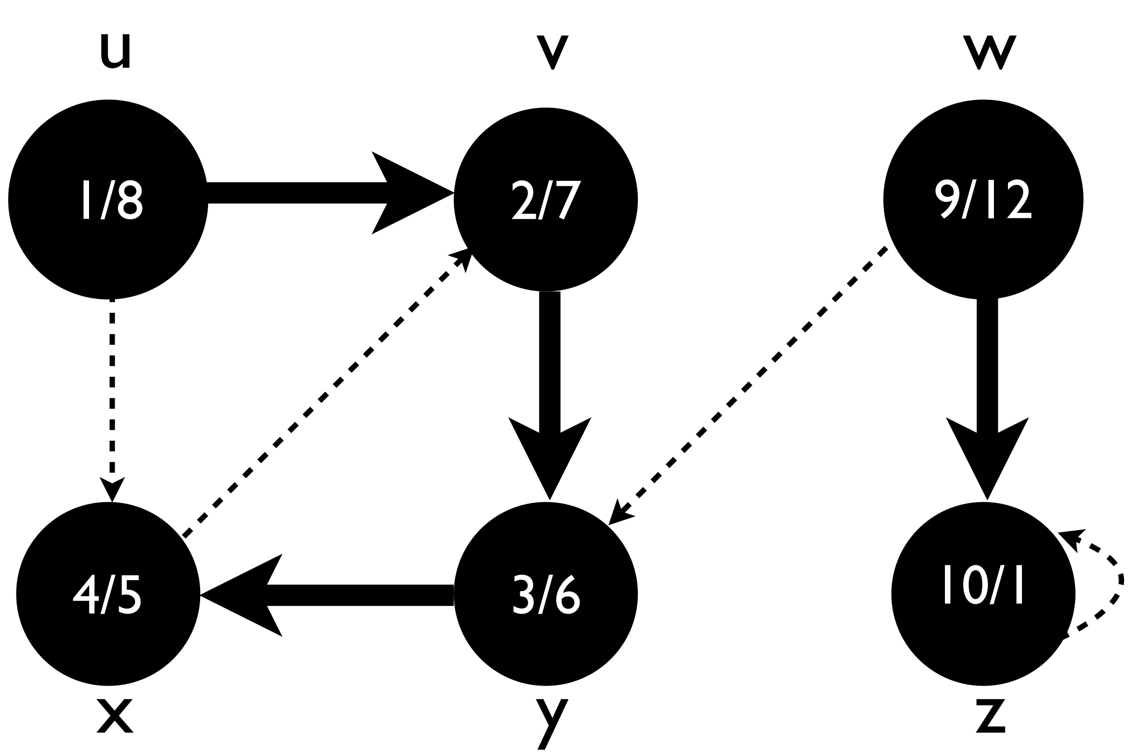


BuscaProf(G)

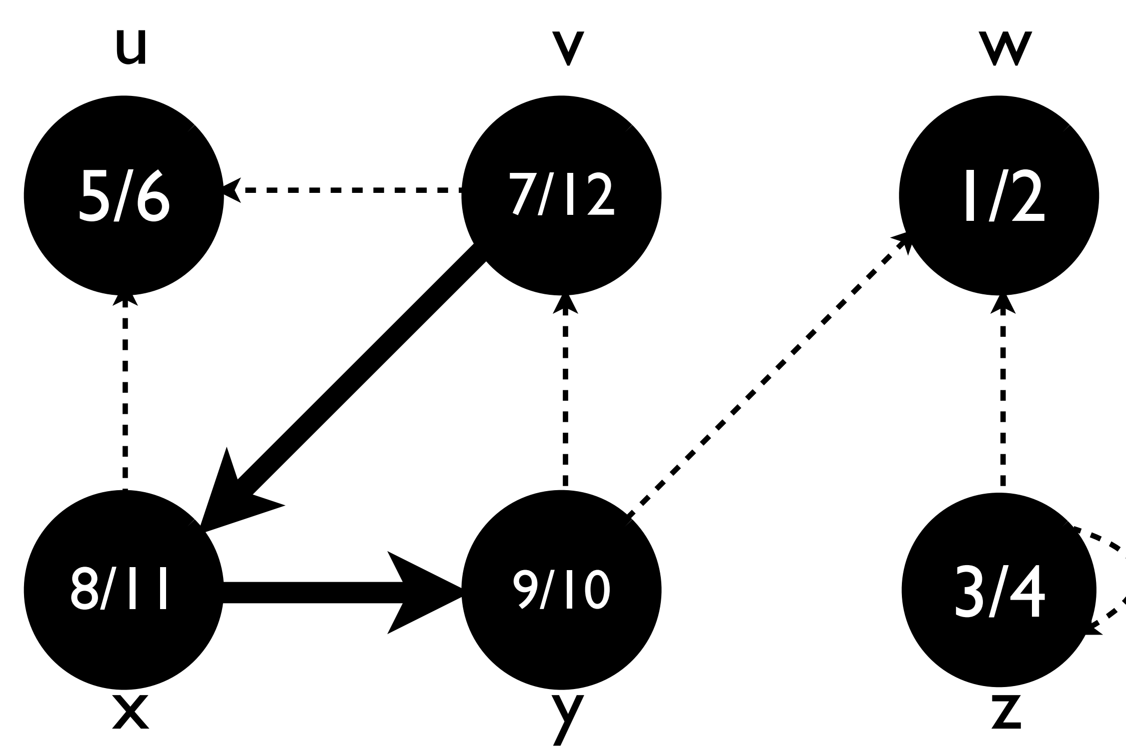


BuscaProf(G^T)

Componentes Conectados

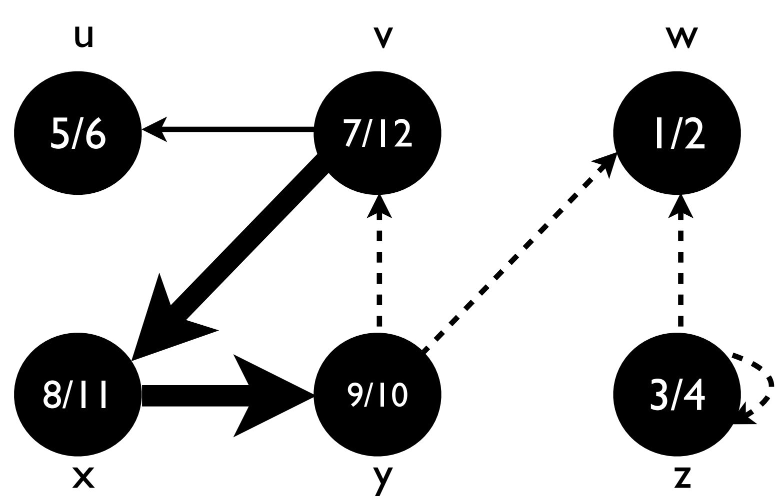


BuscaProf(G)

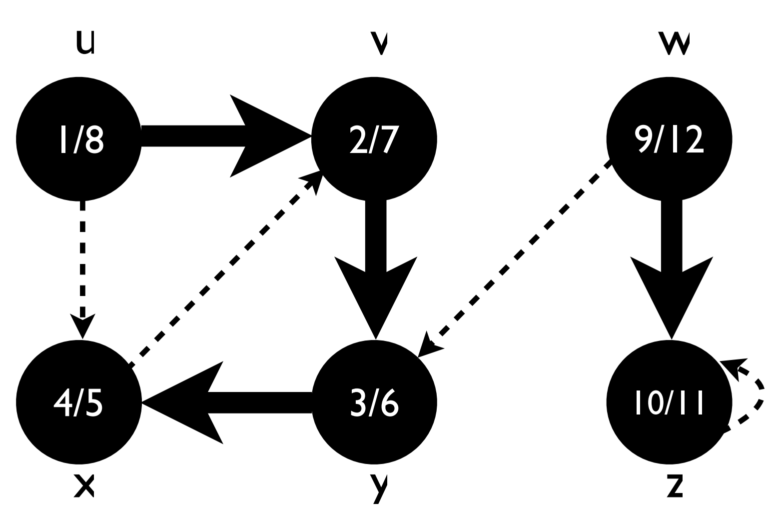


BuscaProf(G^T)

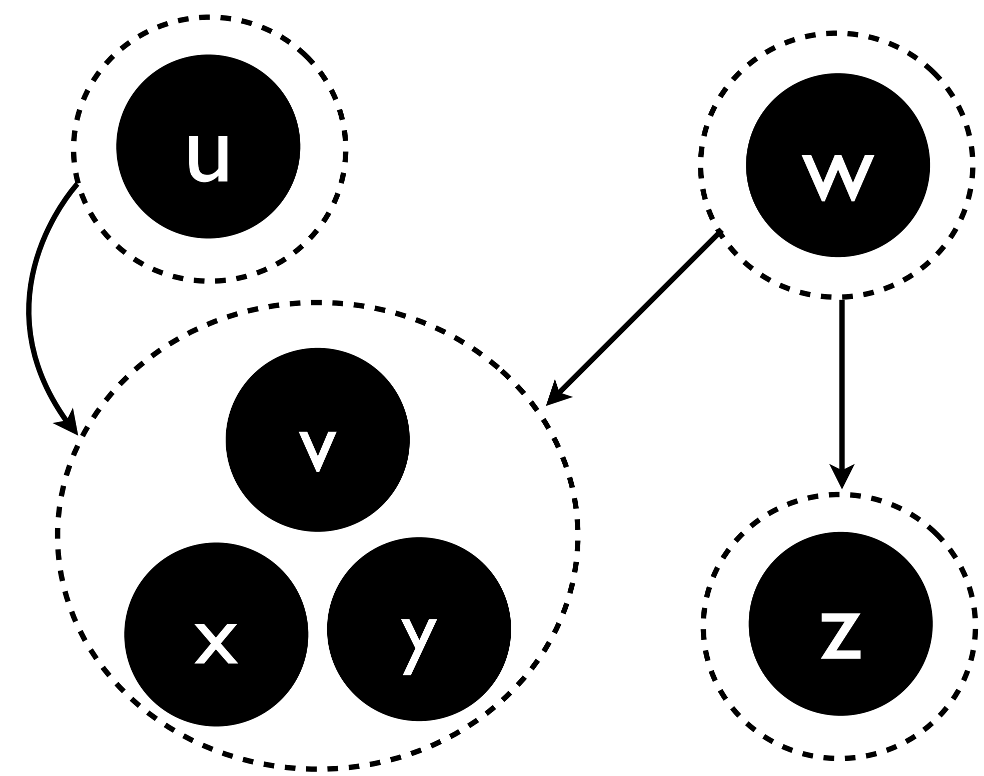
Componentes Conectados



BuscaProf(G^T)



BuscaProf(G)



Resultado

Lição da aula de hoje

Lição da aula de hoje

1. O que aprendemos na aula de hoje?
2. Importância da estrutura de dados grafo
3. Como implementá-la em computador (Listas x Matrizes)
4. Busca em profundidade e complexidade
5. Aplicação da busca em profundidade

Próximos capítulos...

Próximos capítulos...

1. Mais aplicações da busca em profundidade
2. Complexidade e Aplicações
3. Árvores geradoras
4. Caminhos mínimos

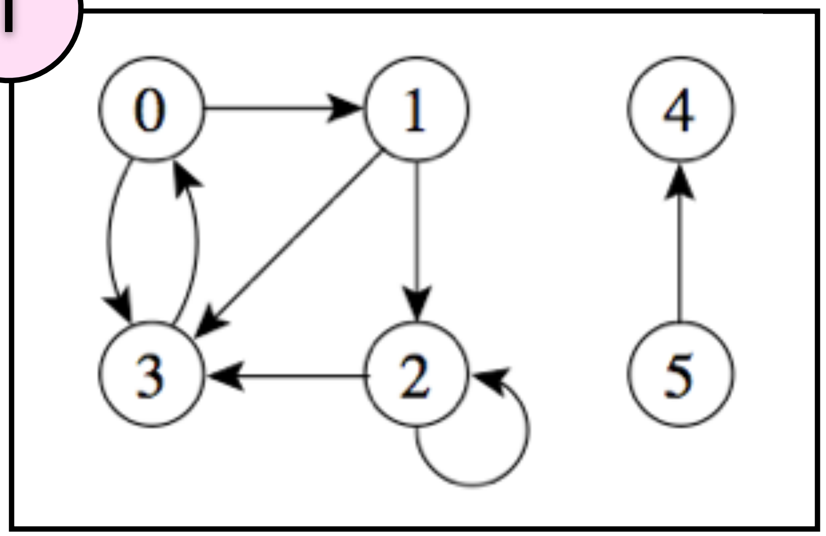
Para casa...

Para casa...

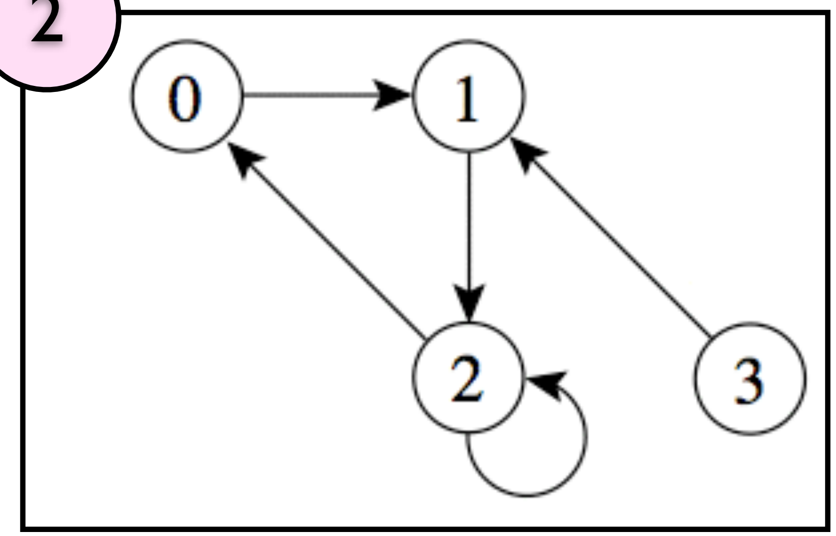
- Para cada um dos grafos a seguir, representá-los como matrizes e listas de adjacências
- Discutir quais vantagens e desvantagens de cada implementação para cada grafo
- Realizar a busca em profundidade em cada grafo
- Realizar ordenação topológica (quando cabível)
- Achar os CPTs fortemente conectados

Para casa...

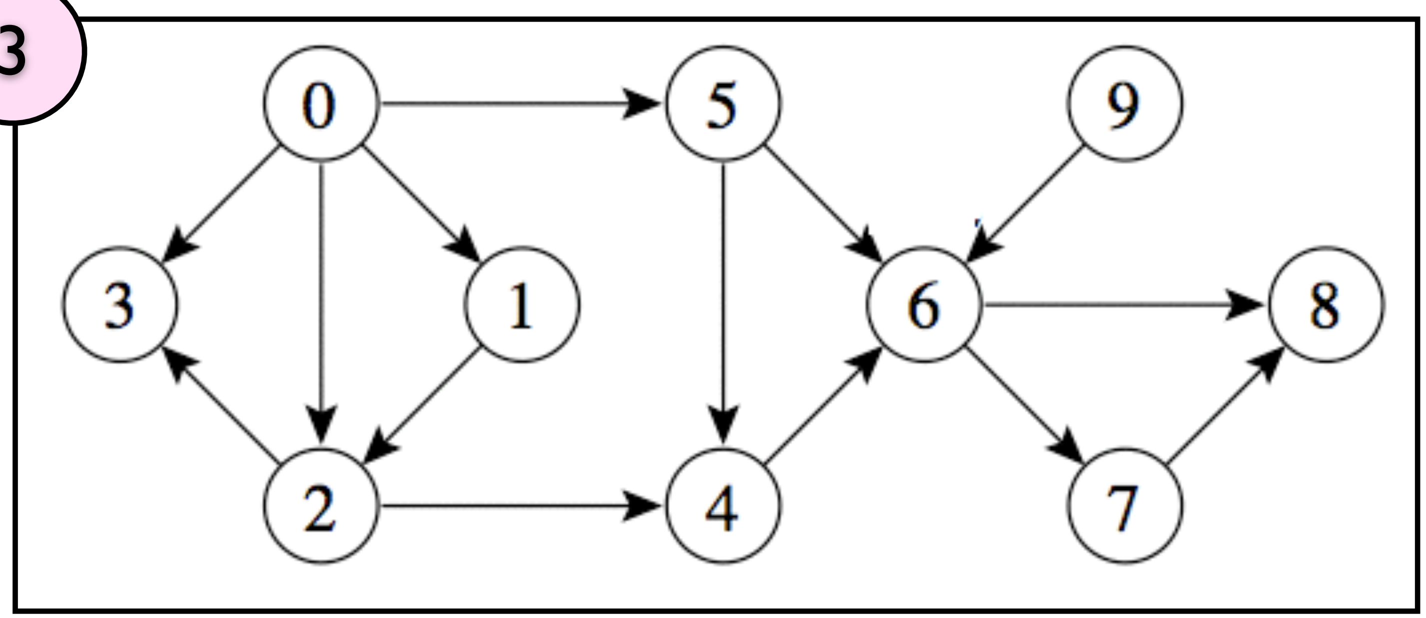
1



2



3



Referências

- *Projeto de Algoritmos com implementações em Java e C++*. Nivio Ziviani, 2007. Cap. 7
- *Introduction to Algorithms*. T. Cormen, C. Leiserson, R. Rivest, and C. Stein, 2nd ed. Caps. 22-26
- *Introduction to Algorithms – A creative approach*. Udi Manber. Cap. 7
- *The Algorithm design manual*. Steven Skiena. Cap. 4

Obrigado!