Algoritmos em Grafos

Ordenação Topológica

Prof. João Caram

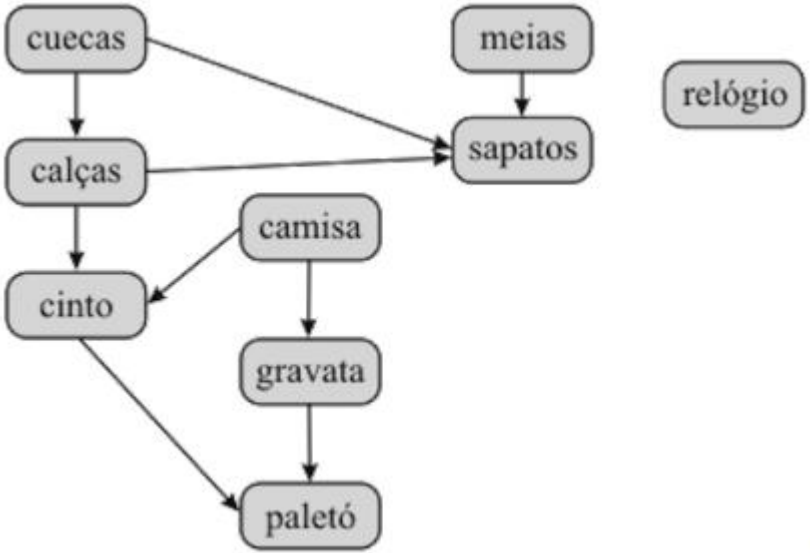
Ordenação topológica

- Dado um DAG, é possível dispor seus vértices de modo que cada vértice apareça antes de todos os seus sucessores?

Ordenação topológica

- Ordenação topológica: ordenação linear de vértices na qual cada vértice precede o conjunto que forma seu fecho transitivo direto.

Ordenação topológica



Teorema

- Seja um digrafo G,
  - Ou ele possui ciclos.
  - Ou ele apresenta uma ou mais ordenações topológicas.

Ordenação topológica - algoritmos

- Existem algoritmos com complexidade linear para determinar uma ordenação topológica de um DAG.
  - Algoritmo de Kahn
  - DFS modificado

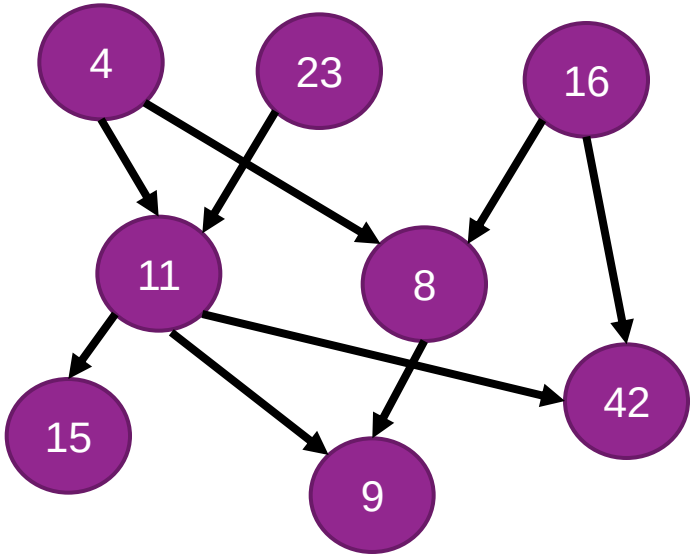
Algoritmo de Kahn (1962)

- Baseado em duas listas:
  - S: conjunto de vértices sem arcos de entrada
  - L: lista de vértices ordenados topologicamente

- Retorna uma lista de ordenação topológica OU detecta a existência de um ciclo

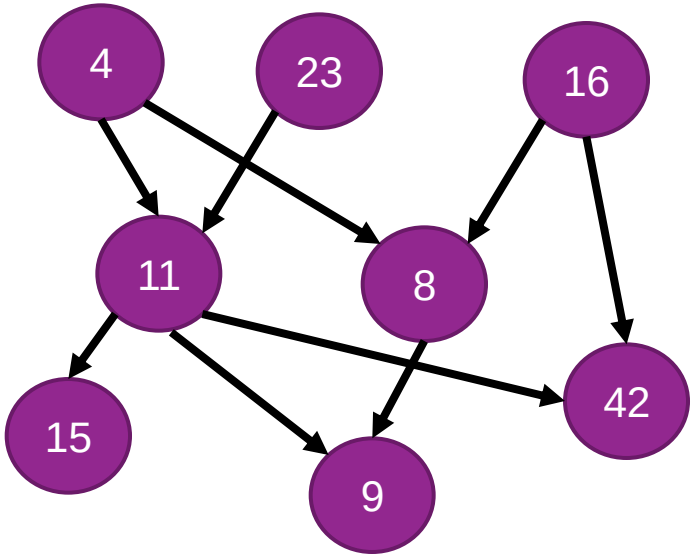
Algoritmo de Kahn

```
L = ∅;
S = todos os vértices sem arcos de entrada;
Enquanto S ≠ ∅ faça
   remover um vértice v de S
   inserir o vértice v em L
   para cada arco v,w existente faça
      remover o arco v,w de E
      se w não possuir mais arcos de entrada
          inserir w em S
   Fim para
Fim enquanto
Se E = ∅ retorna L  //lista ordenada
Senão o grafo possui pelo menos um ciclo
```



Algoritmo de Kahn

```
L = ∅;
S = todos os vértices sem arcos de entrada;
Enquanto S ≠ ∅ faça
   remover um vértice v de S
   inserir o vértice v em L
   para cada arco v,w existente faça
      remover o arco v,w de E
      se w não possuir mais arcos de entrada
          inserir w em S
   Fim para
Fim enquanto
Se E = ∅ retorna L  //lista ordenada
Senão o grafo possui pelo menos um ciclo
```



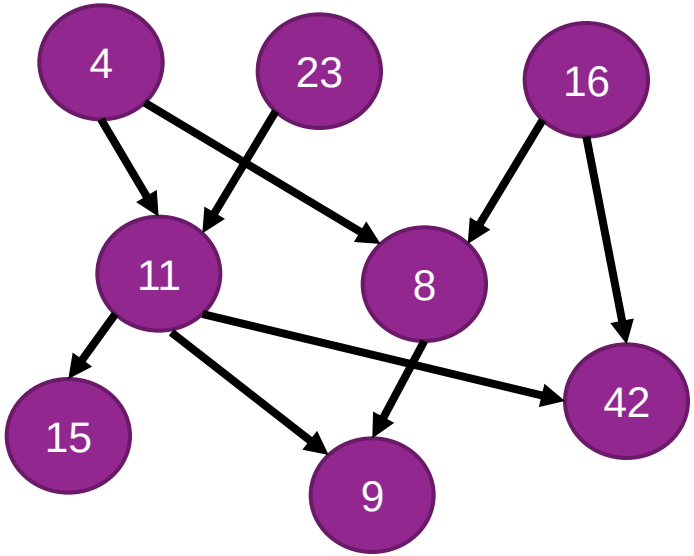
S

4	23	16			

L

Algoritmo de Kahn

```
L = ∅;
S = todos os vértices sem arcos de entrada;
Enquanto S ≠ ∅ faça
   remover um vértice v de S
   inserir o vértice v em L
   para cada arco v,w existente faça
      remover o arco v,w de E
      se w não possuir mais arcos de entrada
          inserir w em S
   Fim para
Fim enquanto
Se E = ∅ retorna L  //lista ordenada
Senão o grafo possui pelo menos um ciclo
```



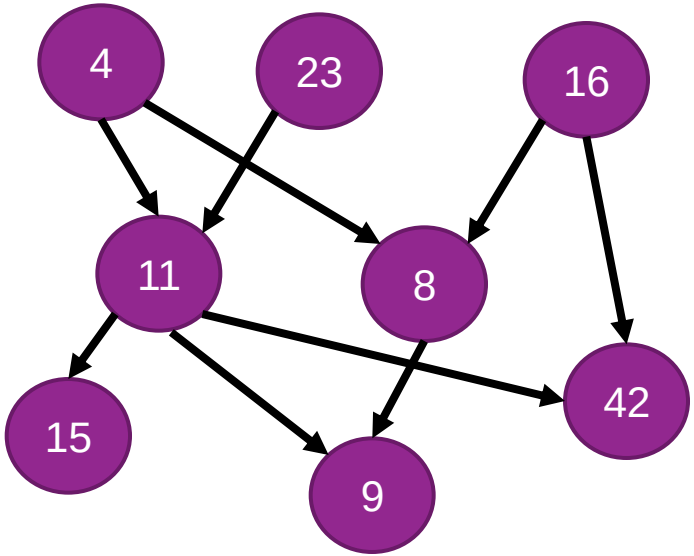
S

4	23	16			

L

Algoritmo de Kahn

```
L = ∅;
S = todos os vértices sem arcos de entrada;
Enquanto S ≠ ∅ faça
   remover um vértice v de S
   inserir o vértice v em L
   para cada arco v,w existente faça
      remover o arco v,w de E
      se w não possuir mais arcos de entrada
          inserir w em S
   Fim para
Fim enquanto
Se E = ∅ retorna L  //lista ordenada
Senão o grafo possui pelo menos um ciclo
```



S

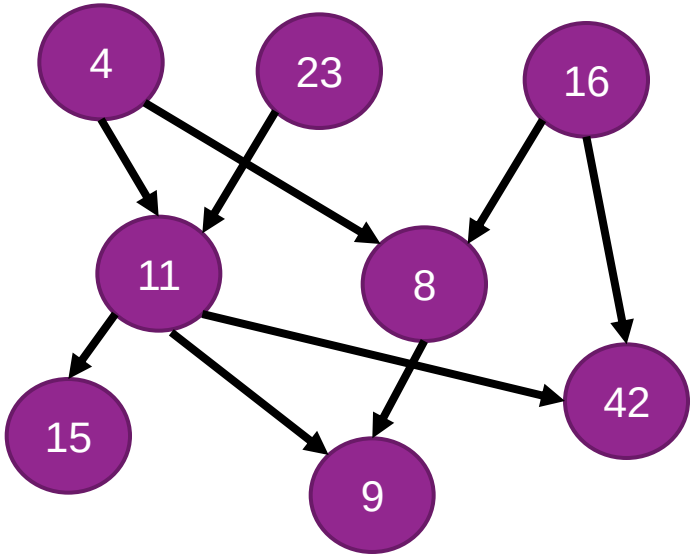
23	16				

L

4					

Algoritmo de Kahn

```
L = ∅;
S = todos os vértices sem arcos de entrada;
Enquanto S ≠ ∅ faça
    remover um vértice v de S
    inserir o vértice v em L
    para cada arco v,w existente faça
        remover o arco v,w de E
        se w não possuir mais arcos de entrada
            inserir w em S
    Fim para
Fim enquanto
Se E = ∅ retorna L  //lista ordenada
Senão o grafo possui pelo menos um ciclo
```



S

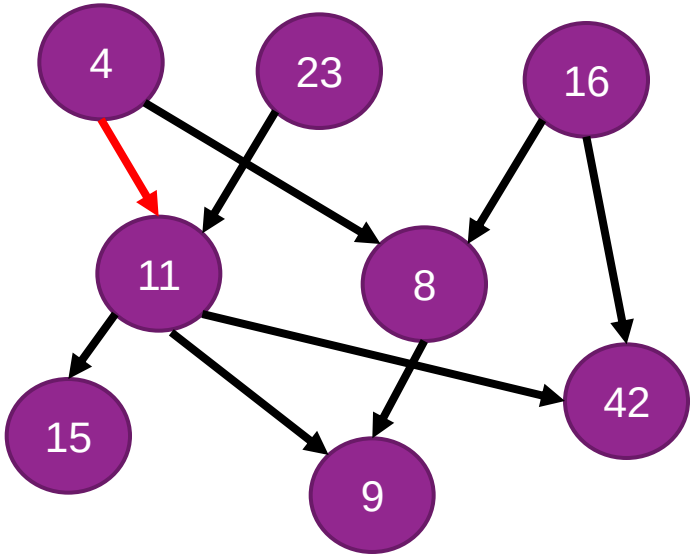
23	16				

L

4					

Algoritmo de Kahn

```
L = ∅;
S = todos os vértices sem arcos de entrada;
Enquanto S ≠ ∅ faça
    remover um vértice v de S
    inserir o vértice v em L
    para cada arco v,w existente faça
        remover o arco v,w de E
        se w não possuir mais arcos de entrada
            inserir w em S
    Fim para
Fim enquanto
Se E = ∅ retorna L  //lista ordenada
Senão o grafo possui pelo menos um ciclo
```



S

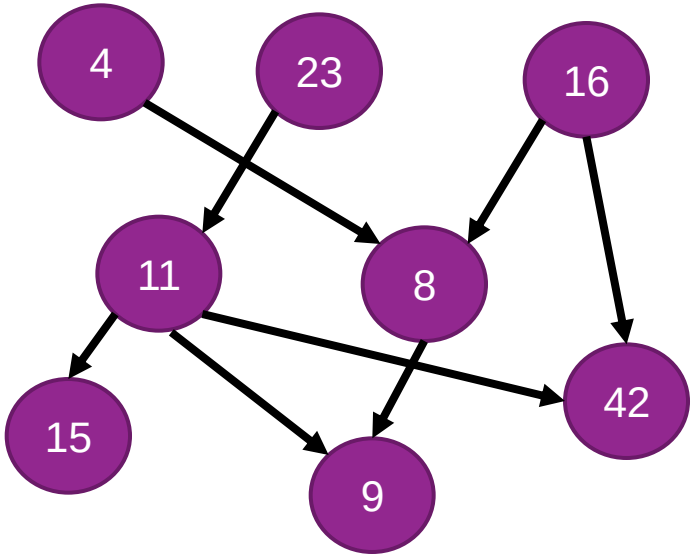
23	16				

L

4					

Algoritmo de Kahn

```
L = ∅;
S = todos os vértices sem arcos de entrada;
Enquanto S ≠ ∅ faça
    remover um vértice v de S
    inserir o vértice v em L
    para cada arco v,w existente faça
        remover o arco v,w de E
        se w não possuir mais arcos de entrada
            inserir w em S
    Fim para
Fim enquanto
Se E = ∅ retorna L  //lista ordenada
Senão o grafo possui pelo menos um ciclo
```



S

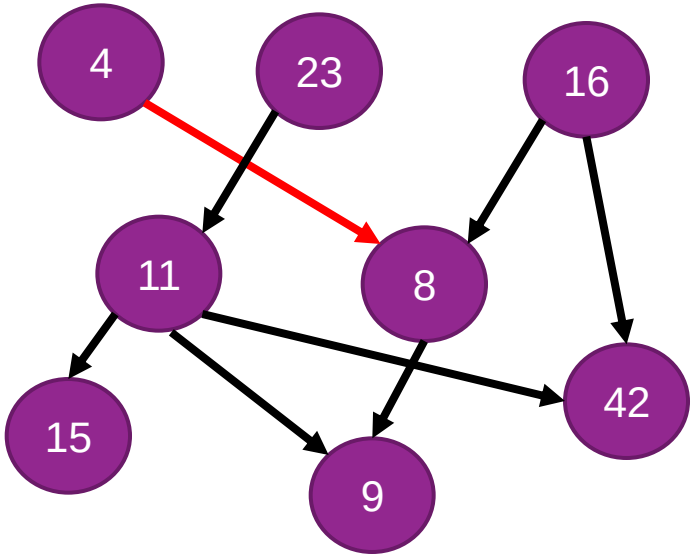
23	16				

L

4					

Algoritmo de Kahn

```
L = ∅;
S = todos os vértices sem arcos de entrada;
Enquanto S ≠ ∅ faça
    remover um vértice v de S
    inserir o vértice v em L
    para cada arco v,w existente faça
        remover o arco v,w de E
        se w não possuir mais arcos de entrada
            inserir w em S
    Fim para
Fim enquanto
Se E = ∅ retorna L  //lista ordenada
Senão o grafo possui pelo menos um ciclo
```



S

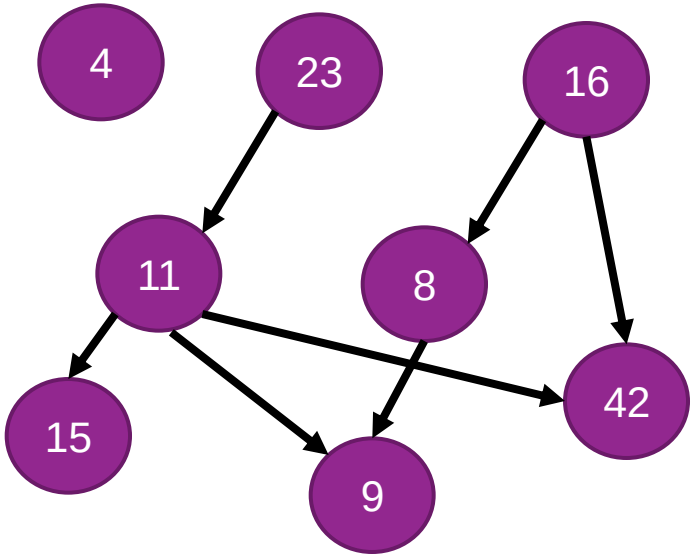
23	16				

L

4					

Algoritmo de Kahn

```
L = ∅;
S = todos os vértices sem arcos de entrada;
Enquanto S ≠ ∅ faça
    remover um vértice v de S
    inserir o vértice v em L
    para cada arco v,w existente faça
        remover o arco v,w de E
        se w não possuir mais arcos de entrada
            inserir w em S
    Fim para
Fim enquanto
Se E = ∅ retorna L  //lista ordenada
Senão o grafo possui pelo menos um ciclo
```



S

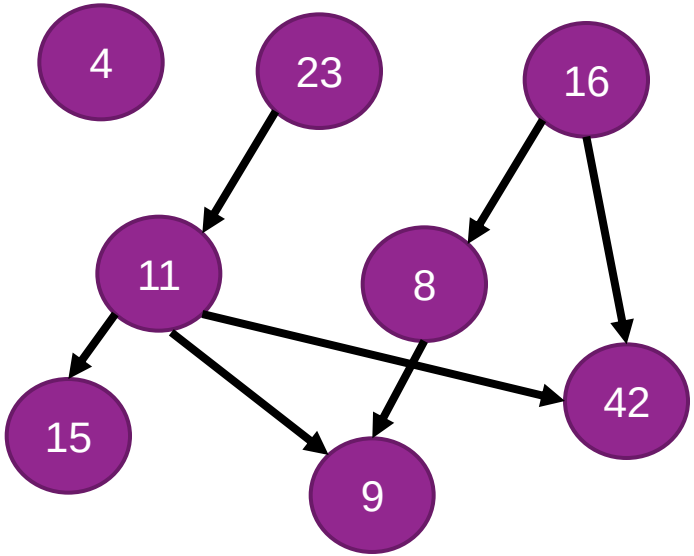
23	16				

L

4					

Algoritmo de Kahn

```
L = ∅;
S = todos os vértices sem arcos de entrada;
Enquanto S ≠ ∅ faça
   remover um vértice v de S
   inserir o vértice v em L
   para cada arco v,w existente faça
      remover o arco v,w de E
      se w não possuir mais arcos de entrada
          inserir w em S
   Fim para
Fim enquanto
Se E = ∅ retorna L  //lista ordenada
Senão o grafo possui pelo menos um ciclo
```



S

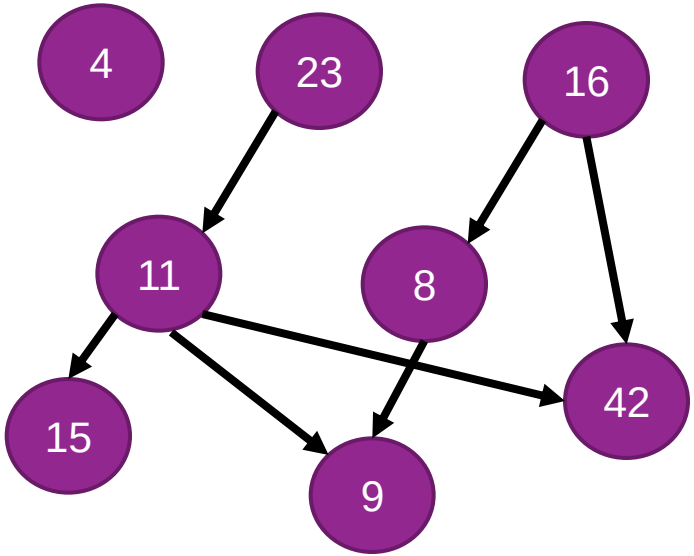
16					

L

4	23				

Algoritmo de Kahn

```
L = ∅;
S = todos os vértices sem arcos de entrada;
Enquanto S ≠ ∅ faça
    remover um vértice v de S
    inserir o vértice v em L
    para cada arco v,w existente faça
        remover o arco v,w de E
        se w não possuir mais arcos de entrada
            inserir w em S
    Fim para
Fim enquanto
Se E = ∅ retorna L  //lista ordenada
Senão o grafo possui pelo menos um ciclo
```



S

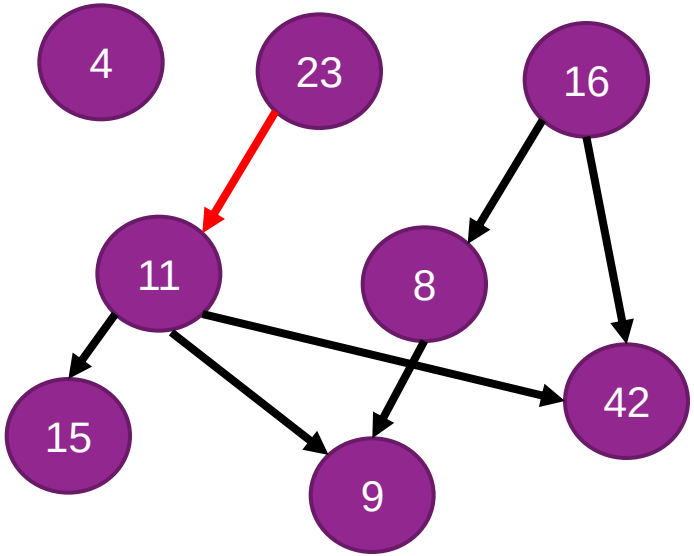
16					

L

4	23				

Algoritmo de Kahn

```
L = ∅;
S = todos os vértices sem arcos de entrada;
Enquanto S ≠ ∅ faça
    remover um vértice v de S
    inserir o vértice v em L
    para cada arco v,w existente faça
        remover o arco v,w de E
        se w não possuir mais arcos de entrada
            inserir w em S
    Fim para
Fim enquanto
Se E = ∅ retorna L  //lista ordenada
Senão o grafo possui pelo menos um ciclo
```



S

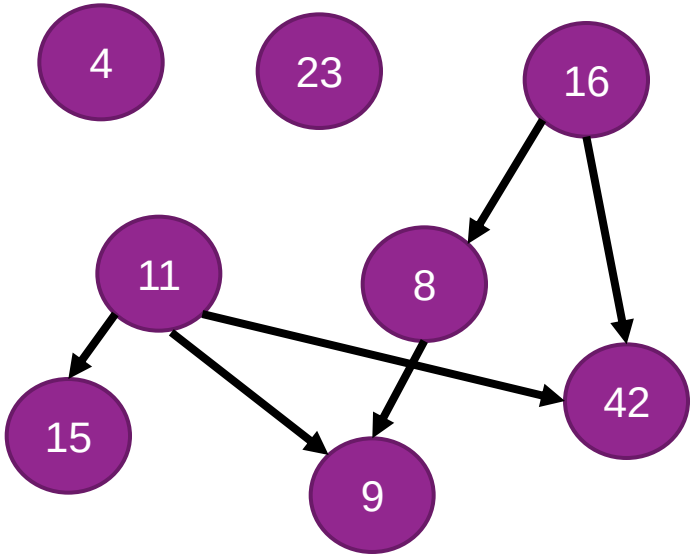
16					

L

4	23				

Algoritmo de Kahn

```
L = ∅;
S = todos os vértices sem arcos de entrada;
Enquanto S ≠ ∅ faça
    remover um vértice v de S
    inserir o vértice v em L
    para cada arco v,w existente faça
        remover o arco v,w de E
        se w não possuir mais arcos de entrada
            inserir w em S
    Fim para
Fim enquanto
Se E = ∅ retorna L  //lista ordenada
Senão o grafo possui pelo menos um ciclo
```



S

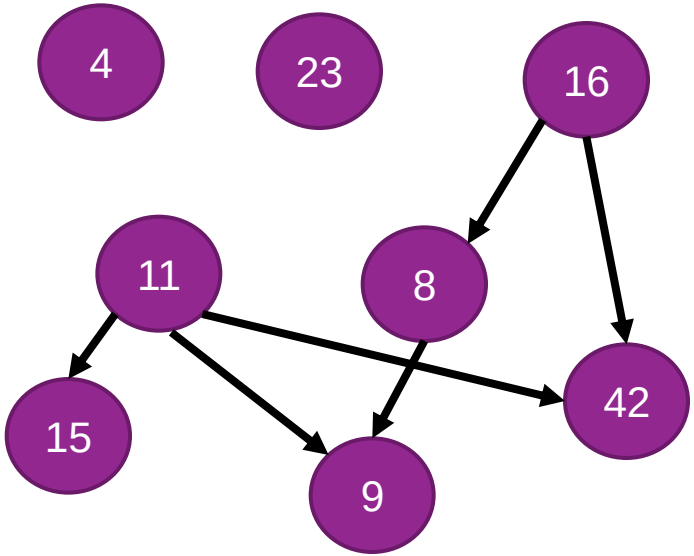
16	11				

L

4	23				

Algoritmo de Kahn

```
L = ∅;
S = todos os vértices sem arcos de entrada;
Enquanto S ≠ ∅ faça
   remover um vértice v de S
   inserir o vértice v em L
   para cada arco v,w existente faça
      remover o arco v,w de E
      se w não possuir mais arcos de entrada
          inserir w em S
   Fim para
Fim enquanto
Se E = ∅ retorna L  //lista ordenada
Senão o grafo possui pelo menos um ciclo
```



S

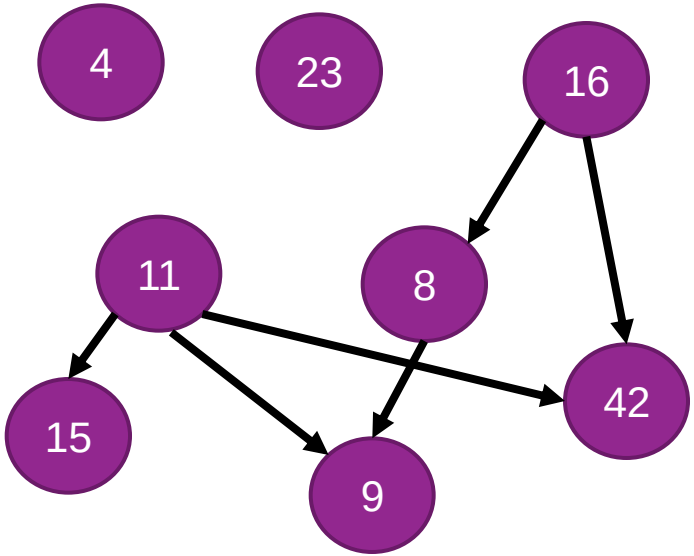
11					

L

4	23	16			

Algoritmo de Kahn

```
L = ∅;
S = todos os vértices sem arcos de entrada;
Enquanto S ≠ ∅ faça
   remover um vértice v de S
   inserir o vértice v em L
   para cada arco v,w existente faça
      remover o arco v,w de E
      se w não possuir mais arcos de entrada
         inserir w em S
   Fim para
Fim enquanto
Se E = ∅ retorna L  //lista ordenada
Senão o grafo possui pelo menos um ciclo
```



S

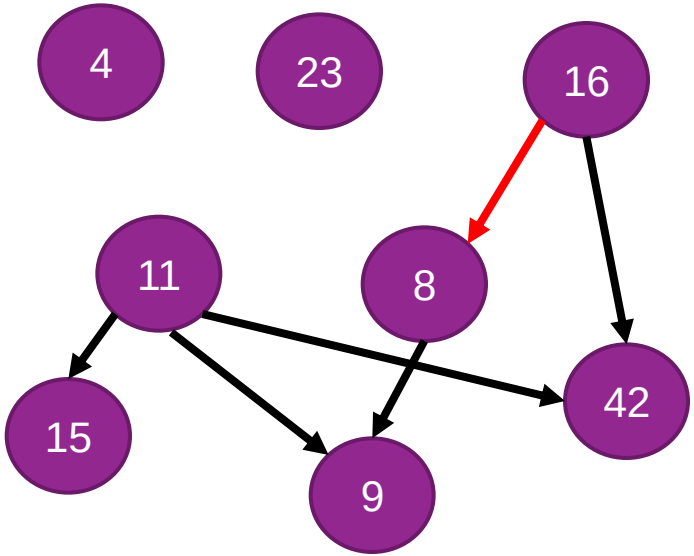
11					

L

4	23	16			

Algoritmo de Kahn

```
L = ∅;
S = todos os vértices sem arcos de entrada;
Enquanto S ≠ ∅ faça
    remover um vértice v de S
    inserir o vértice v em L
    para cada arco v,w existente faça
        remover o arco v,w de E
        se w não possuir mais arcos de entrada
            inserir w em S
    Fim para
Fim enquanto
Se E = ∅ retorna L  //lista ordenada
Senão o grafo possui pelo menos um ciclo
```



S

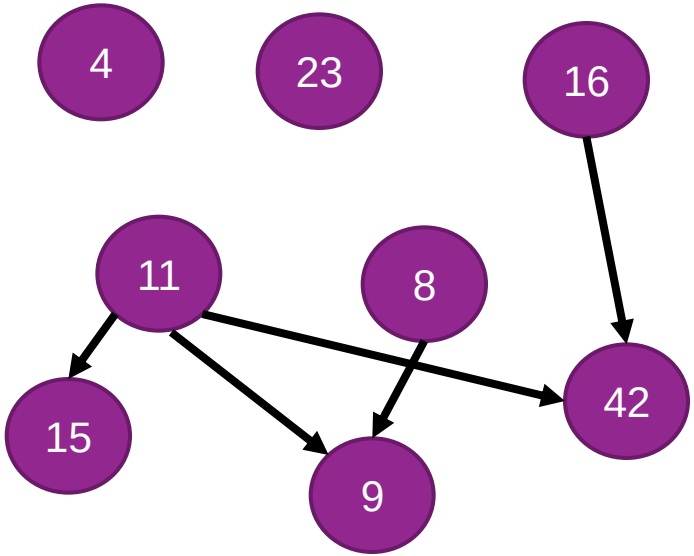
11					

L

4	23	16			

Algoritmo de Kahn

```
L = ∅;
S = todos os vértices sem arcos de entrada;
Enquanto S ≠ ∅ faça
   remover um vértice v de S
   inserir o vértice v em L
   para cada arco v,w existente faça
      remover o arco v,w de E
      se w não possuir mais arcos de entrada
         inserir w em S
   Fim para
Fim enquanto
Se E = ∅ retorna L  //lista ordenada
Senão o grafo possui pelo menos um ciclo
```



S

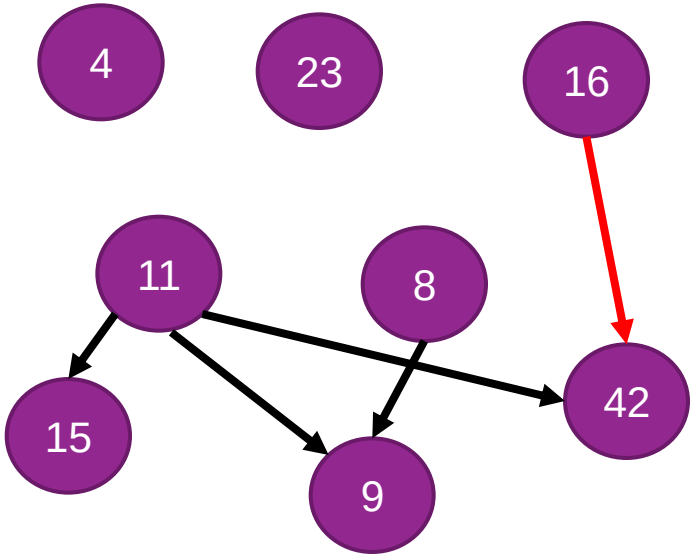
11	8				

L

4	23	16			

Algoritmo de Kahn

```
L = ∅;
S = todos os vértices sem arcos de entrada;
Enquanto S ≠ ∅ faça
    remover um vértice v de S
    inserir o vértice v em L
    para cada arco v,w existente faça
        remover o arco v,w de E
        se w não possuir mais arcos de entrada
            inserir w em S
    Fim para
Fim enquanto
Se E = ∅ retorna L  //lista ordenada
Senão o grafo possui pelo menos um ciclo
```



S

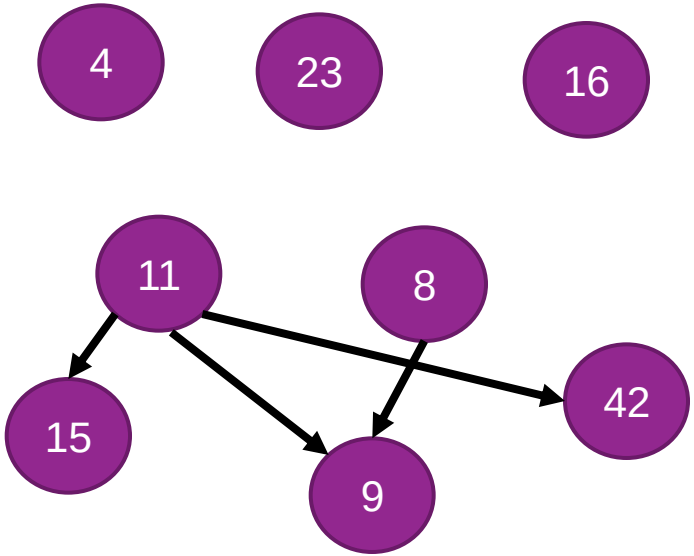
11	8				

L

4	23	16			

Algoritmo de Kahn

```
L = ∅;
S = todos os vértices sem arcos de entrada;
Enquanto S ≠ ∅ faça
   remover um vértice v de S
   inserir o vértice v em L
   para cada arco v,w existente faça
      remover o arco v,w de E
      se w não possuir mais arcos de entrada
         inserir w em S
   Fim para
Fim enquanto
Se E = ∅ retorna L  //lista ordenada
Senão o grafo possui pelo menos um ciclo
```



S

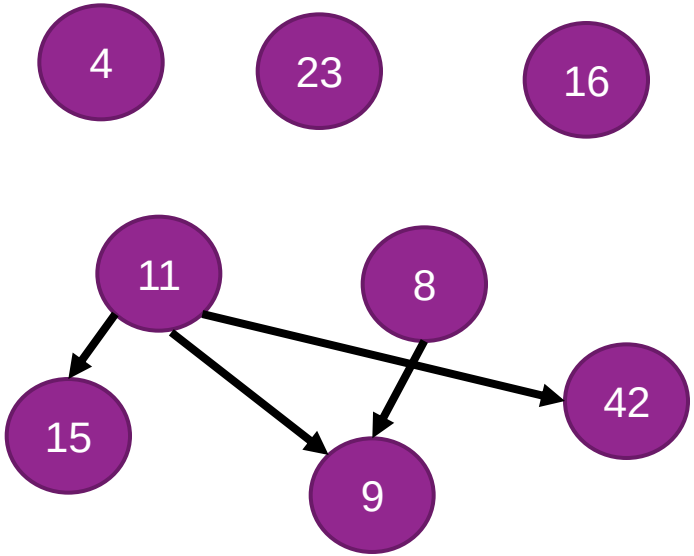
11	8				

L

4	23	16			

Algoritmo de Kahn

```
L = ∅;
S = todos os vértices sem arcos de entrada;
Enquanto S ≠ ∅ faça
   remover um vértice v de S
   inserir o vértice v em L
   para cada arco v,w existente faça
      remover o arco v,w de E
      se w não possuir mais arcos de entrada
          inserir w em S
   Fim para
Fim enquanto
Se E = ∅ retorna L  //lista ordenada
Senão o grafo possui pelo menos um ciclo
```



S

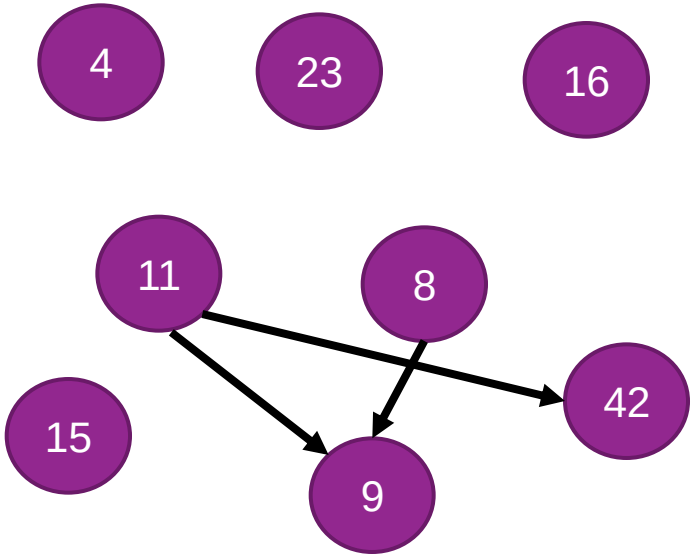
8					

L

4	23	16	11		

Algoritmo de Kahn

```
L = ∅;
S = todos os vértices sem arcos de entrada;
Enquanto S ≠ ∅ faça
   remover um vértice v de S
   inserir o vértice v em L
   para cada arco v,w existente faça
      remover o arco v,w de E
      se w não possuir mais arcos de entrada
         inserir w em S
   Fim para
Fim enquanto
Se E = ∅ retorna L  //lista ordenada
Senão o grafo possui pelo menos um ciclo
```



S

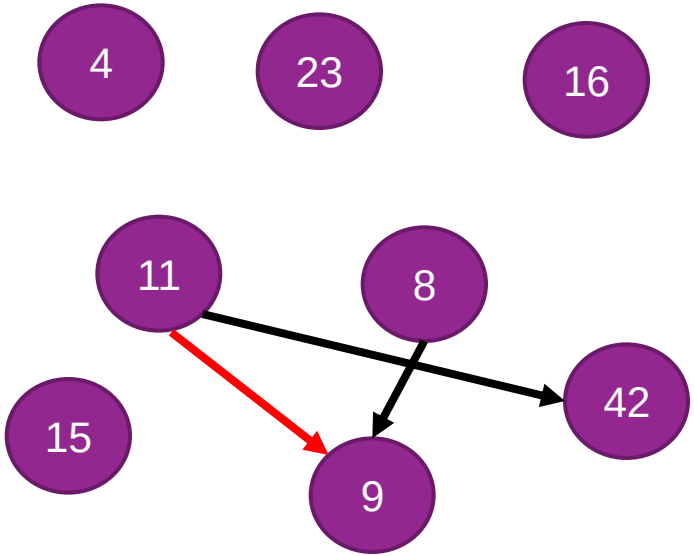
8	15				

L

4	23	16	11		

Algoritmo de Kahn

```
L = ∅;
S = todos os vértices sem arcos de entrada;
Enquanto S ≠ ∅ faça
    remover um vértice v de S
    inserir o vértice v em L
    para cada arco v,w existente faça
        remover o arco v,w de E
        se w não possuir mais arcos de entrada
            inserir w em S
    Fim para
Fim enquanto
Se E = ∅ retorna L  //lista ordenada
Senão o grafo possui pelo menos um ciclo
```



S

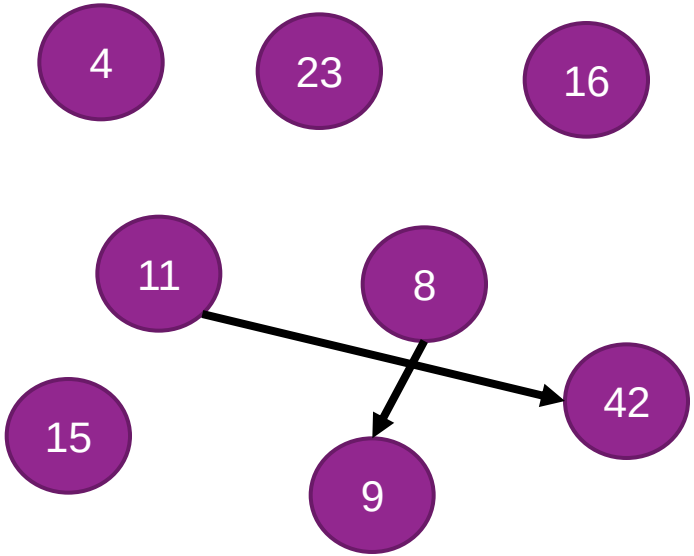
8	15				

L

4	23	16	11		

Algoritmo de Kahn

```
L = ∅;
S = todos os vértices sem arcos de entrada;
Enquanto S ≠ ∅ faça
   remover um vértice v de S
   inserir o vértice v em L
   para cada arco v,w existente faça
      remover o arco v,w de E
      se w não possuir mais arcos de entrada
         inserir w em S
   Fim para
Fim enquanto
Se E = ∅ retorna L  //lista ordenada
Senão o grafo possui pelo menos um ciclo
```



S

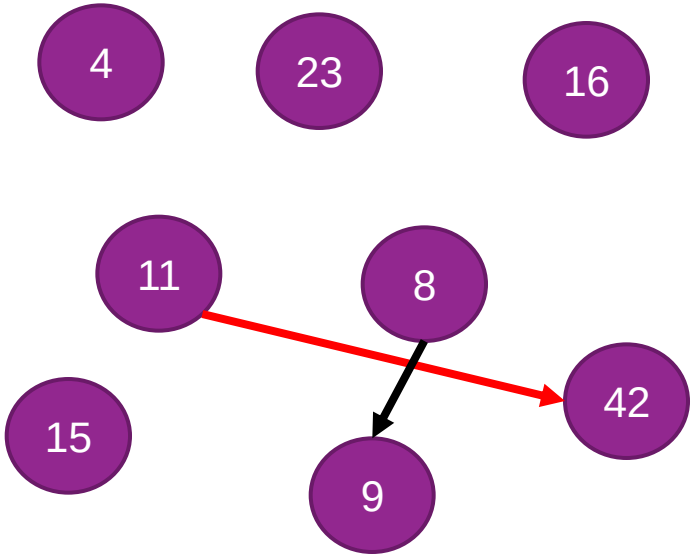
8	15				

L

4	23	16	11		

Algoritmo de Kahn

```
L = ∅;
S = todos os vértices sem arcos de entrada;
Enquanto S ≠ ∅ faça
    remover um vértice v de S
    inserir o vértice v em L
    para cada arco v,w existente faça
        remover o arco v,w de E
        se w não possuir mais arcos de entrada
            inserir w em S
    Fim para
Fim enquanto
Se E = ∅ retorna L  //lista ordenada
Senão o grafo possui pelo menos um ciclo
```



S

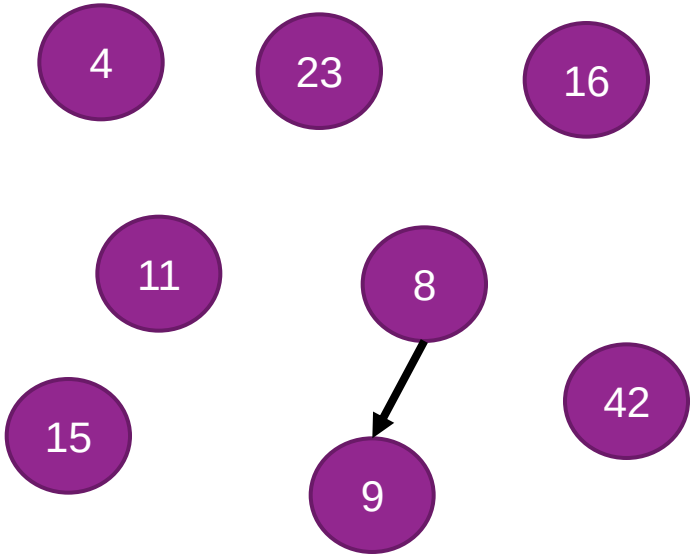
8	15				

L

4	23	16	11		

Algoritmo de Kahn

```
L = ∅;
S = todos os vértices sem arcos de entrada;
Enquanto S ≠ ∅ faça
   remover um vértice v de S
   inserir o vértice v em L
   para cada arco v,w existente faça
      remover o arco v,w de E
      se w não possuir mais arcos de entrada
         inserir w em S
   Fim para
Fim enquanto
Se E = ∅ retorna L  //lista ordenada
Senão o grafo possui pelo menos um ciclo
```



S

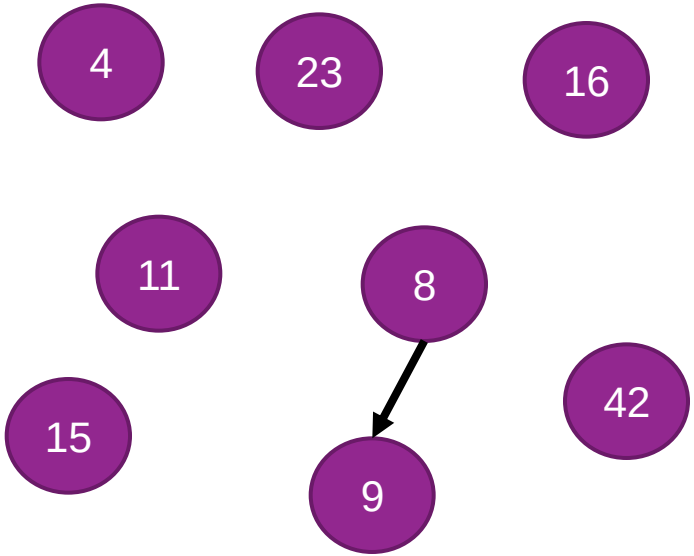
8	15	42			

L

4	23	16	11		

Algoritmo de Kahn

```
L = ∅;
S = todos os vértices sem arcos de entrada;
Enquanto S ≠ ∅ faça
   remover um vértice v de S
   inserir o vértice v em L
   para cada arco v,w existente faça
      remover o arco v,w de E
      se w não possuir mais arcos de entrada
          inserir w em S
   Fim para
Fim enquanto
Se E = ∅ retorna L  //lista ordenada
Senão o grafo possui pelo menos um ciclo
```



S

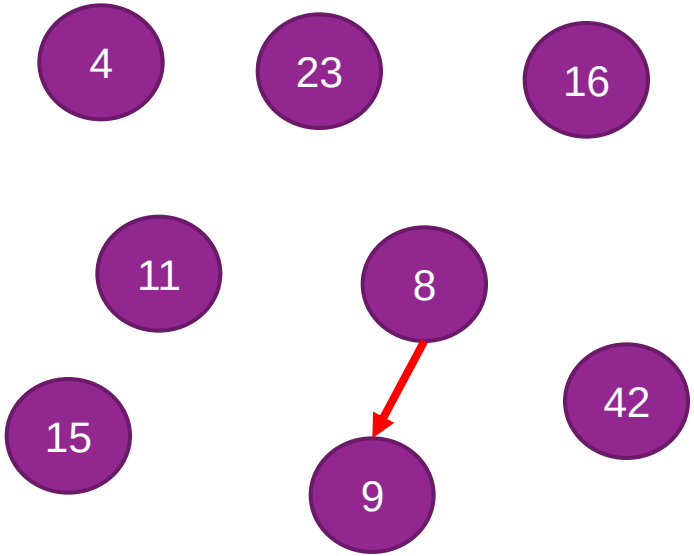
15	42				

L

4	23	16	11	8	

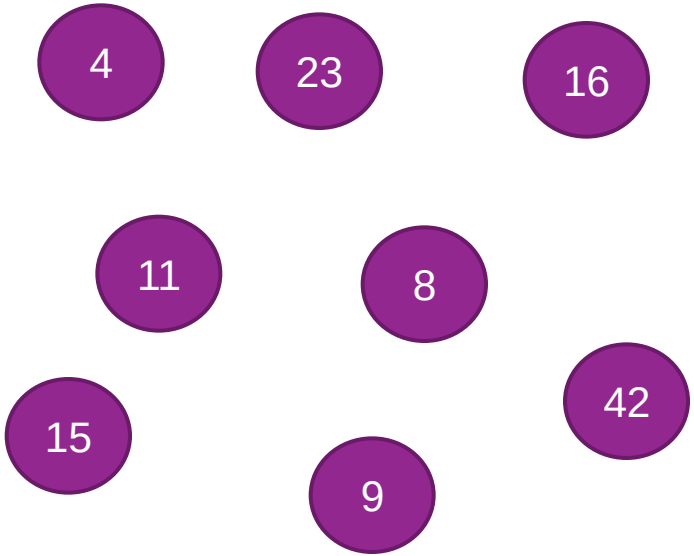
Algoritmo de Kahn

```
L = ∅;
S = todos os vértices sem arcos de entrada;
Enquanto S ≠ ∅ faça
   remover um vértice v de S
   inserir o vértice v em L
   para cada arco v,w existente faça
      remover o arco v,w de E
      se w não possuir mais arcos de entrada
         inserir w em S
   Fim para
Fim enquanto
Se E = ∅ retorna L  //lista ordenada
Senão o grafo possui pelo menos um ciclo
```



Algoritmo de Kahn

```
L = ∅;
S = todos os vértices sem arcos de entrada;
Enquanto S ≠ ∅ faça
   remover um vértice v de S
   inserir o vértice v em L
   para cada arco v,w existente faça
      remover o arco v,w de E
      se w não possuir mais arcos de entrada
         inserir w em S
   Fim para
Fim enquanto
Se E = ∅ retorna L  //lista ordenada
Senão o grafo possui pelo menos um ciclo
```



S

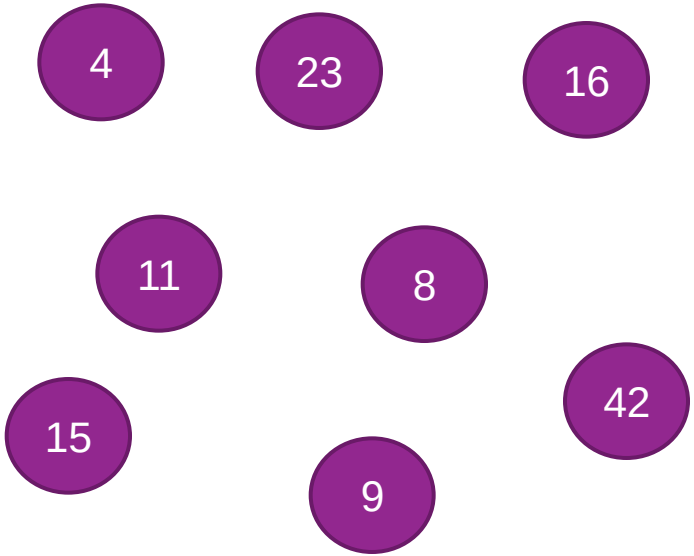
15	42	9			

L

4	23	16	11	8	

Algoritmo de Kahn

```
L = ∅;
S = todos os vértices sem arcos de entrada;
Enquanto S ≠ ∅ faça
   remover um vértice v de S
   inserir o vértice v em L
   para cada arco v,w existente faça
      remover o arco v,w de E
      se w não possuir mais arcos de entrada
         inserir w em S
   Fim para
Fim enquanto
Se E = ∅ retorna L  //lista ordenada
Senão o grafo possui pelo menos um ciclo
```



S

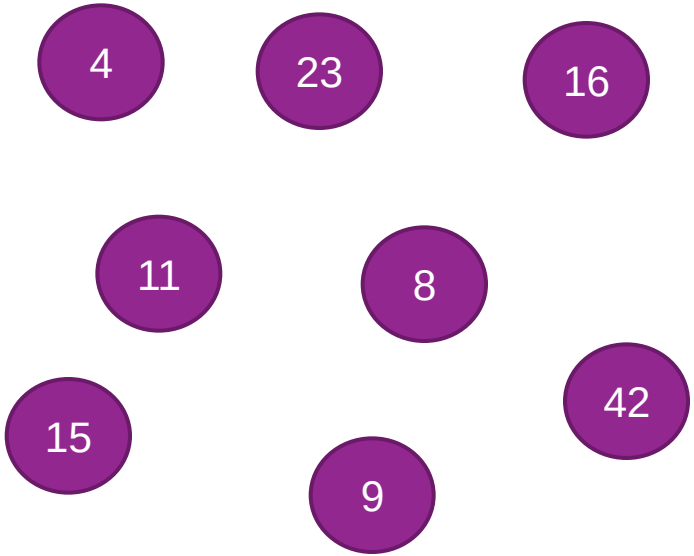
15	42	9			

L

4	23	16	11	8	

Algoritmo de Kahn

```
L = ∅;
S = todos os vértices sem arcos de entrada;
Enquanto S ≠ ∅ faça
    remover um vértice v de S
    inserir o vértice v em L
    para cada arco v,w existente faça
        remover o arco v,w de E
        se w não possuir mais arcos de entrada
            inserir w em S
    Fim para
Fim enquanto
Se E = ∅ retorna L  //lista ordenada
Senão o grafo possui pelo menos um ciclo
```



S

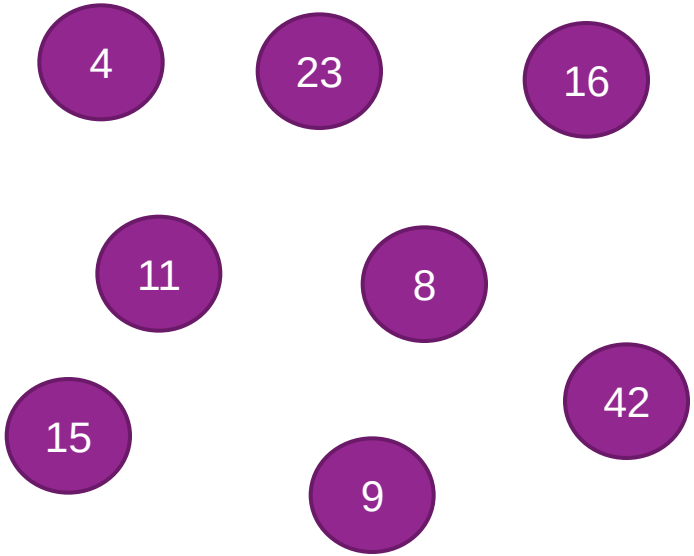
42	9				

L

4	23	16	11	8	15

Algoritmo de Kahn

```
L = ∅;
S = todos os vértices sem arcos de entrada;
Enquanto S ≠ ∅ faça
   remover um vértice v de S
   inserir o vértice v em L
   para cada arco v,w existente faça
      remover o arco v,w de E
      se w não possuir mais arcos de entrada
         inserir w em S
   Fim para
Fim enquanto
Se E = ∅ retorna L  //lista ordenada
Senão o grafo possui pelo menos um ciclo
```



S

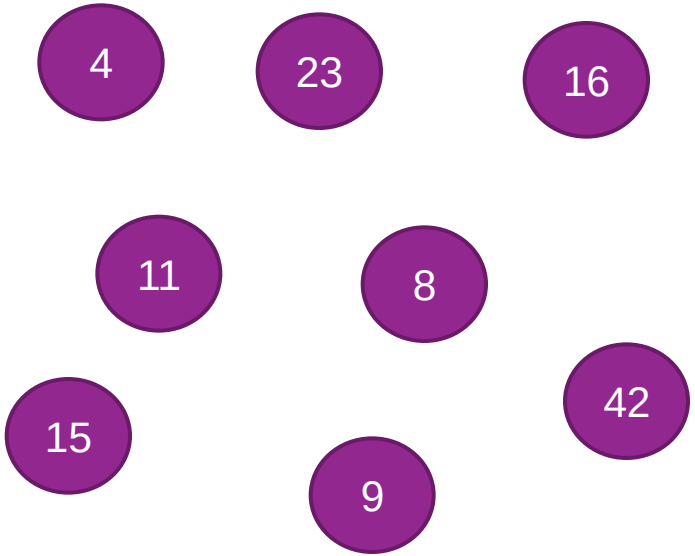
9					

L

4	23	16	11	8	15	42

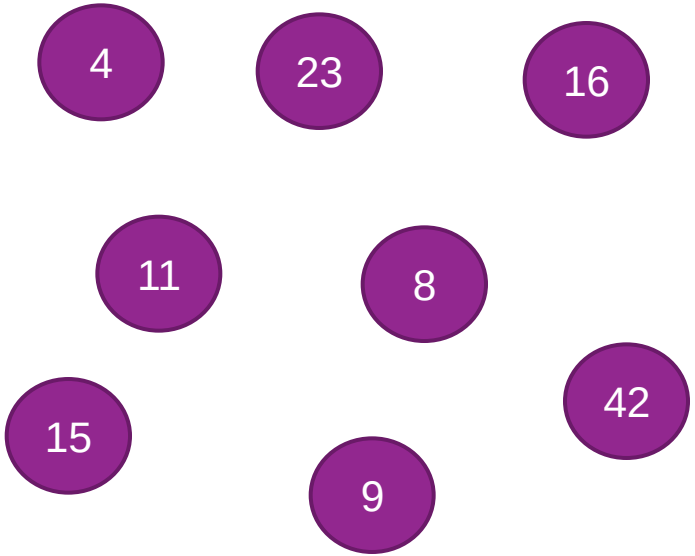
Algoritmo de Kahn

```
L = ∅;
S = todos os vértices sem arcos de entrada;
Enquanto S ≠ ∅ faça
   remover um vértice v de S
   inserir o vértice v em L
   para cada arco v,w existente faça
      remover o arco v,w de E
      se w não possuir mais arcos de entrada
         inserir w em S
   Fim para
Fim enquanto
Se E = ∅ retorna L  //lista ordenada
Senão o grafo possui pelo menos um ciclo
```



Algoritmo de Kahn

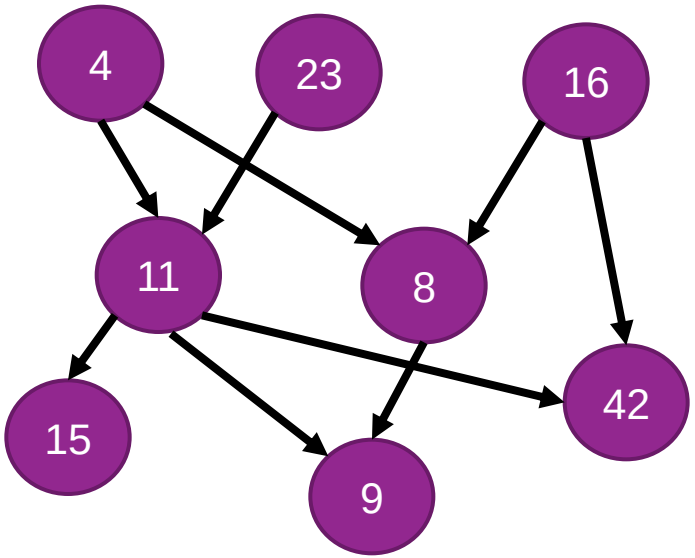
```
L = ∅;
S = todos os vértices sem arcos de entrada;
Enquanto S ≠ ∅ faça
   remover um vértice v de S
   inserir o vértice v em L
   para cada arco v,w existente faça
      remover o arco v,w de E
      se w não possuir mais arcos de entrada
          inserir w em S
   Fim para
Fim enquanto
Se E = ∅ retorna L  //lista ordenada
Senão o grafo possui pelo menos um ciclo
```



Algoritmo de Kahn

L

4	23	16	11	8	15	42	9



DFS e ordenação

- Podemos utilizar uma versão levemente alterada da busca em profundidade para realizar a ordenação topológica

- Basta, ao finalizar um vértice preto, inseri-lo no início de uma lista L

Algoritmo DFS – principal (visita)

```
timestamp = timestamp + 1;
u.descoberta = timestamp;
u.cor = cinza;
Para cada vértice v vizinho de u faça
        se v.cor == branco
                v.pai = u;
                Visitar(v);
        Fim se
Fim Para
u.cor = preto;
timestamp = timestamp+1;
u.término = timestamp;
```

Algoritmo DFS – principal (visita)

```
u.cor = cinza;
Para cada vértice v vizinho de u faça
        se v.cor == branco

                Visitar(v);
        Fim se
Fim Para
u.cor = preto;
```

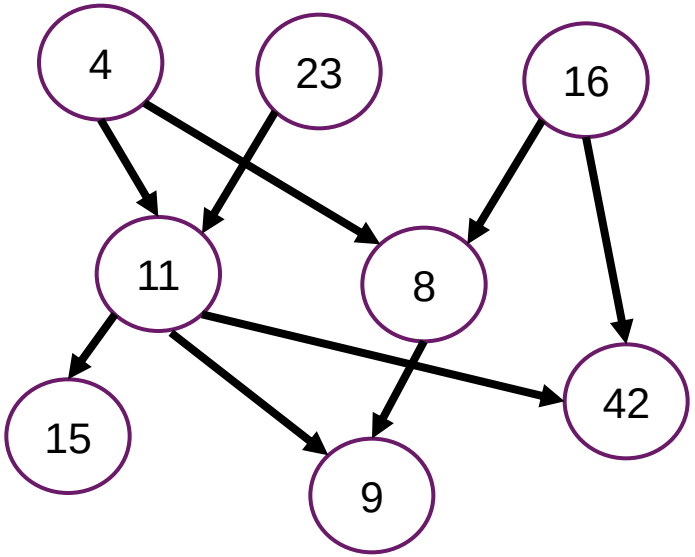
Algoritmo DFS – principal (visita)

```
u.cor = cinza;
Para cada vértice v vizinho de u faça
        se v.cor == branco

                Visitar(v);
        Fim se
Fim Para
u.cor = preto;
L.inserirInicio(u)
```

Algoritmo DFS – principal (visita)

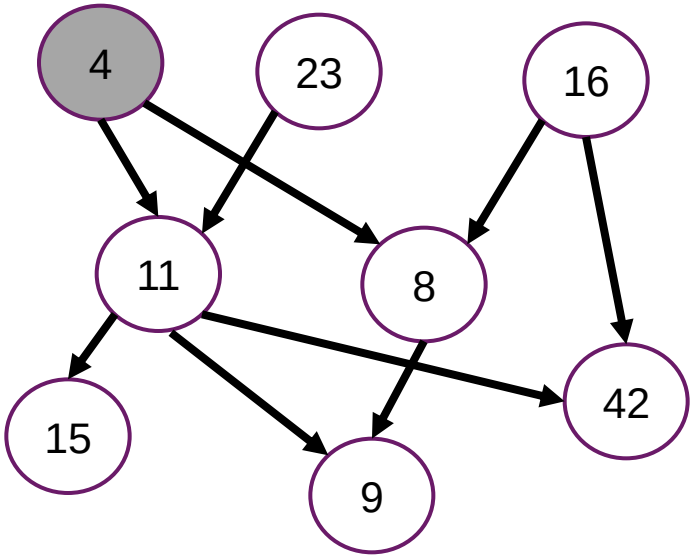
```
u.cor = cinza;
Para cada vértice v vizinho de u faça
        se v.cor == branco
                Visitar(v);
        Fim se
Fim Para
u.cor = preto;
L.inserirInicio(u);
```



L

Algoritmo DFS – principal (visita)

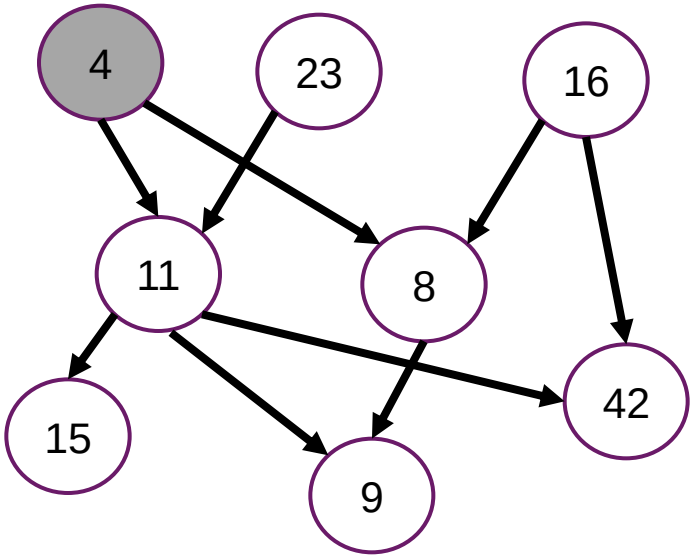
```
u.cor = cinza;
Para cada vértice v vizinho de u faça
        se v.cor == branco
                Visitar(v);
        Fim se
Fim Para
u.cor = preto;
L.inserirInicio(u);
```



L

Algoritmo DFS – principal (visita)

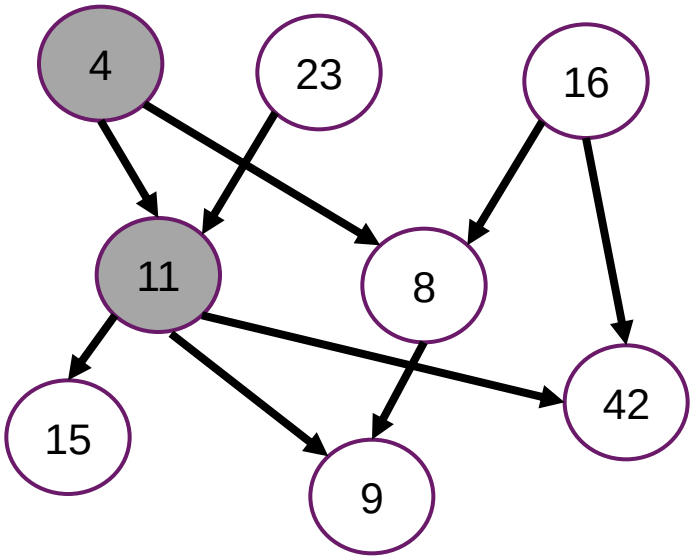
```
u.cor = cinza;
Para cada vértice v vizinho de u faça
        se v.cor == branco
                Visitar(v);
        Fim se
Fim Para
u.cor = preto;
L.inserirInicio(u);
```



L

Algoritmo DFS – principal (visita)

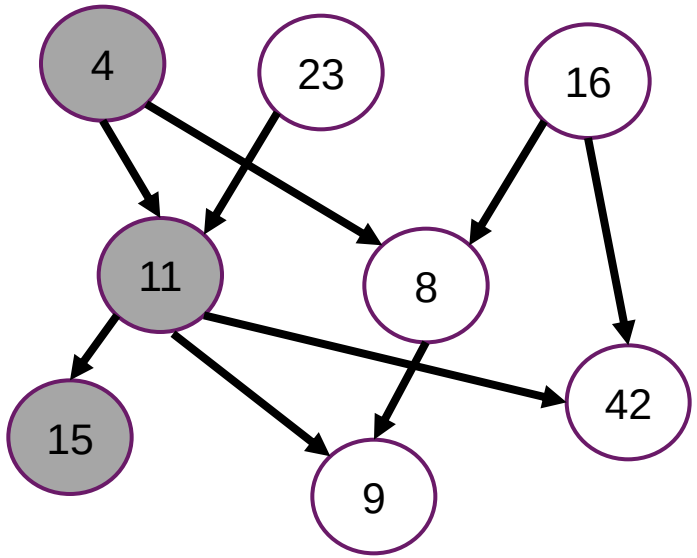
```
u.cor = cinza;
Para cada vértice v vizinho de u faça
        se v.cor == branco
                Visitar(v);
        Fim se
Fim Para
u.cor = preto;
L.inserirInicio(u);
```



L

Algoritmo DFS – principal (visita)

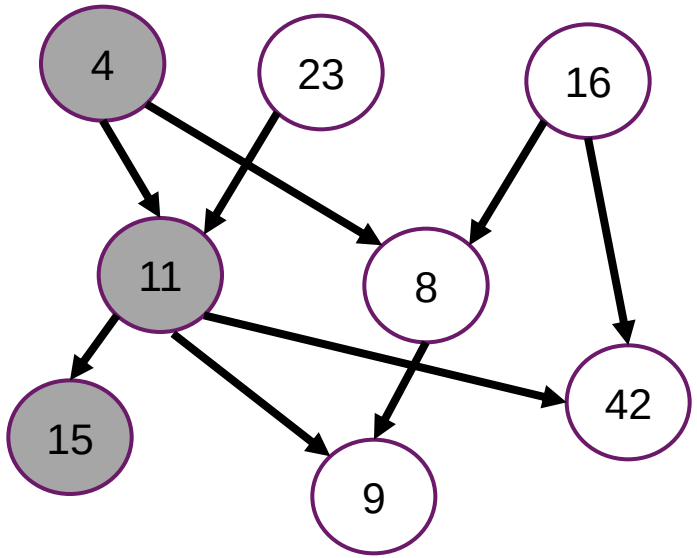
```
u.cor = cinza;
Para cada vértice v vizinho de u faça
        se v.cor == branco
                Visitar(v);
        Fim se
Fim Para
u.cor = preto;
L.inserirInicio(u);
```



L

Algoritmo DFS – principal (visita)

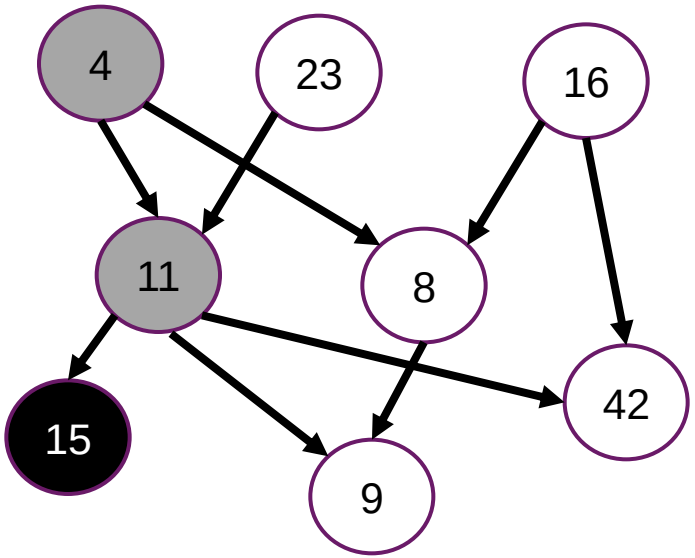
```
u.cor = cinza;
Para cada vértice v vizinho de u faça
        se v.cor == branco
                Visitar(v);
        Fim se
Fim Para
u.cor = preto;
L.inserirInicio(u);
```



L

Algoritmo DFS – principal (visita)

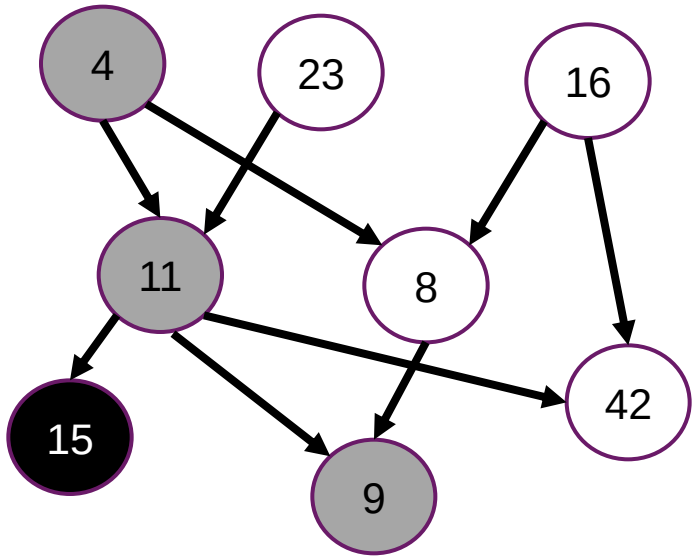
```
u.cor = cinza;
Para cada vértice v vizinho de u faça
        se v.cor == branco
                Visitar(v);
        Fim se
Fim Para
u.cor = preto;
L.inserirInicio(u);
```



L

Algoritmo DFS – principal (visita)

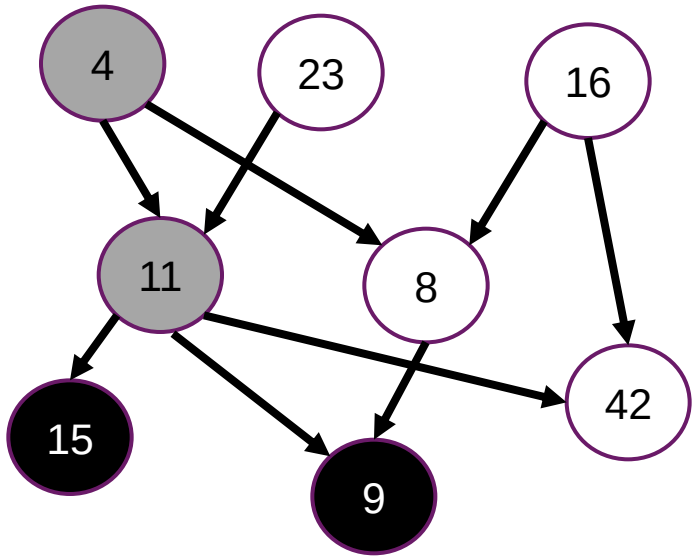
```
u.cor = cinza;
Para cada vértice v vizinho de u faça
        se v.cor == branco
                Visitar(v);
        Fim se
Fim Para
u.cor = preto;
L.inserirInicio(u);
```



L

Algoritmo DFS – principal (visita)

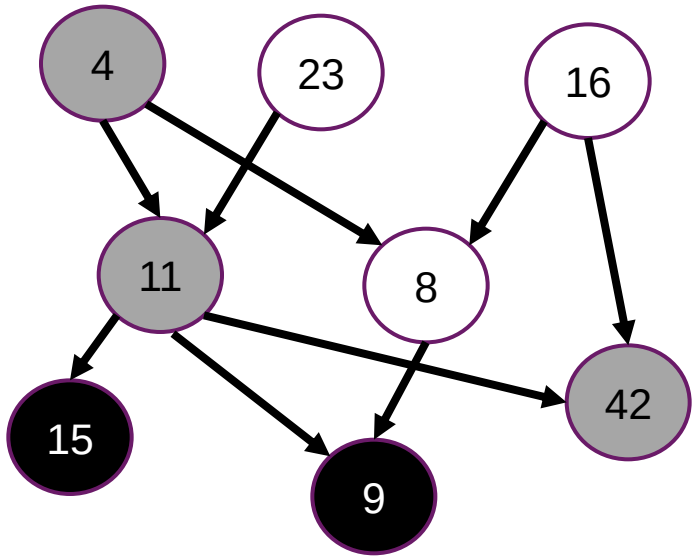
```
u.cor = cinza;
Para cada vértice v vizinho de u faça
        se v.cor == branco
                Visitar(v);
        Fim se
Fim Para
u.cor = preto;
L.inserirInicio(u);
```



L

Algoritmo DFS – principal (visita)

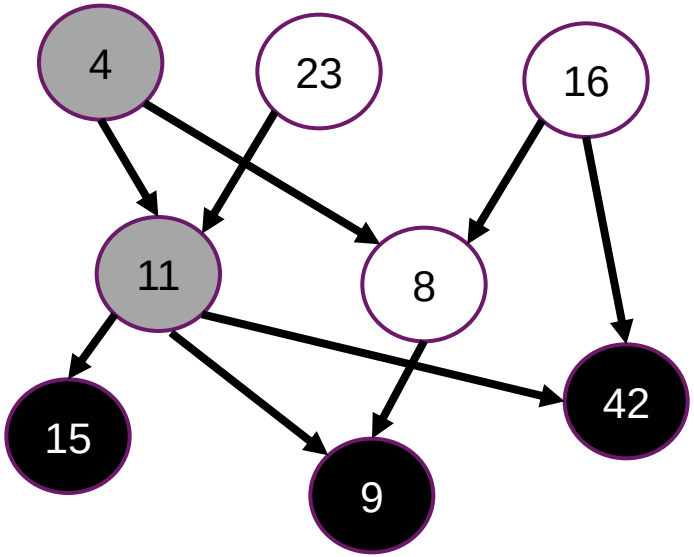
```
u.cor = cinza;
Para cada vértice v vizinho de u faça
        se v.cor == branco
                Visitar(v);
        Fim se
Fim Para
u.cor = preto;
L.inserirInicio(u);
```



L

Algoritmo DFS – principal (visita)

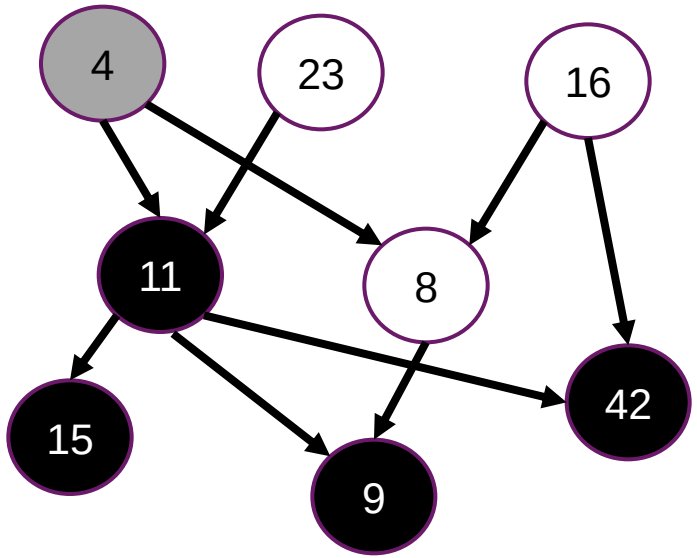
```
u.cor = cinza;
Para cada vértice v vizinho de u faça
        se v.cor == branco
                Visitar(v);
        Fim se
Fim Para
u.cor = preto;
L.inserirInicio(u);
```



L

Algoritmo DFS – principal (visita)

```
u.cor = cinza;
Para cada vértice v vizinho de u faça
        se v.cor == branco
                Visitar(v);
        Fim se
Fim Para
u.cor = preto;
L.inserirInicio(u);
```

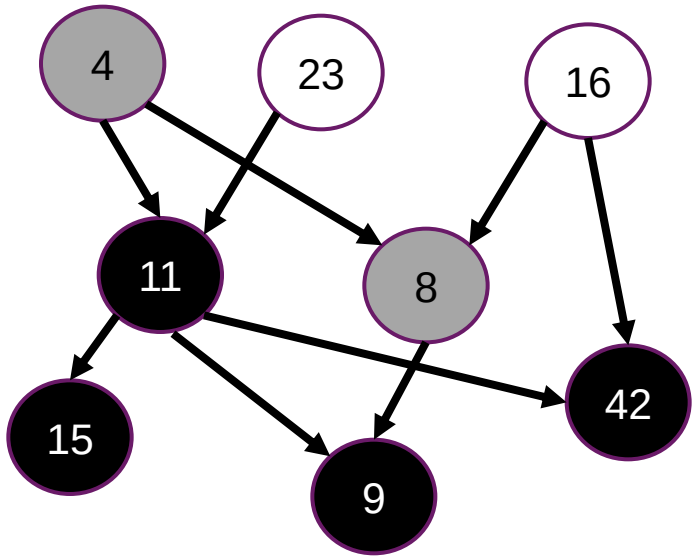


L

11	42	9	15				

Algoritmo DFS – principal (visita)

```
u.cor = cinza;
Para cada vértice v vizinho de u faça
        se v.cor == branco
                Visitar(v);
        Fim se
Fim Para
u.cor = preto;
L.inserirInicio(u);
```

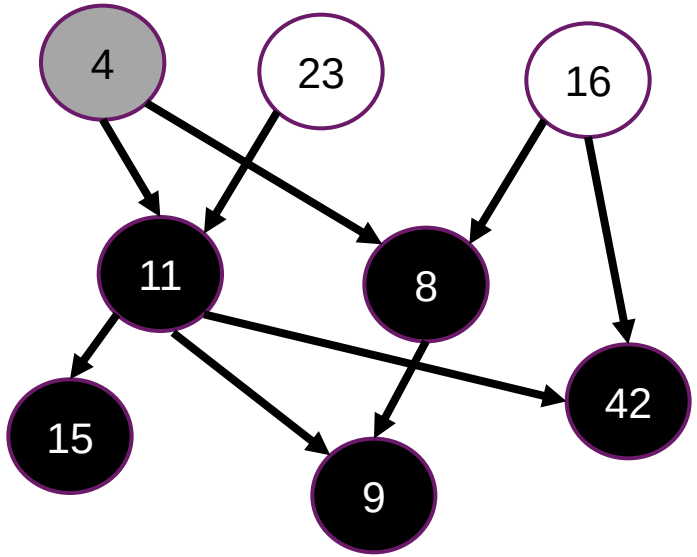


L

11	42	9	15				

Algoritmo DFS – principal (visita)

```
u.cor = cinza;
Para cada vértice v vizinho de u faça
        se v.cor == branco
                Visitar(v);
        Fim se
Fim Para
u.cor = preto;
L.inserirInicio(u);
```

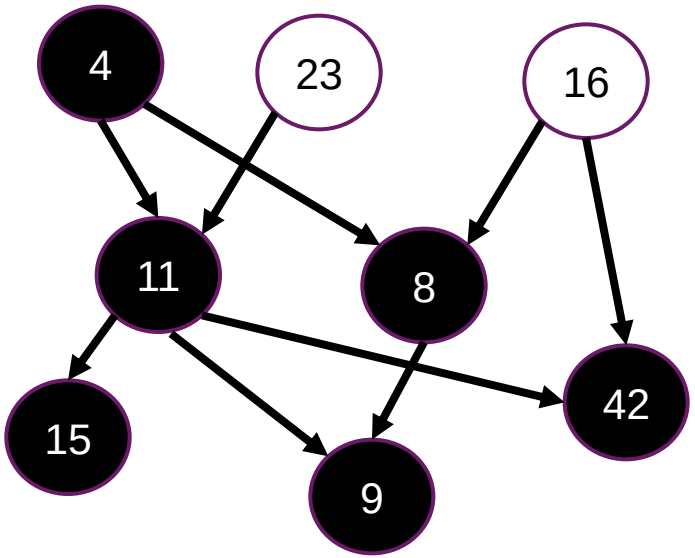


L

8	11	42	9	15			

Algoritmo DFS – principal (visita)

```
u.cor = cinza;
Para cada vértice v vizinho de u faça
        se v.cor == branco
                Visitar(v);
        Fim se
Fim Para
u.cor = preto;
L.inserirInicio(u);
```

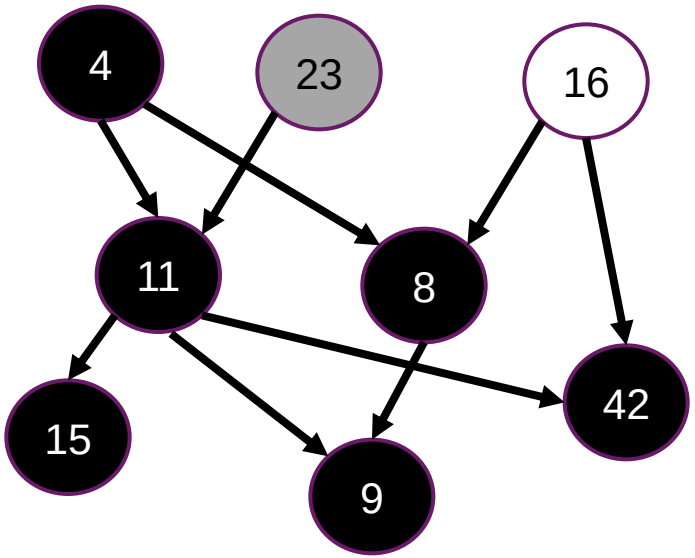


L

4	8	11	42	9	15		

Algoritmo DFS – principal (visita)

```
u.cor = cinza;
Para cada vértice v vizinho de u faça
        se v.cor == branco
                Visitar(v);
        Fim se
Fim Para
u.cor = preto;
L.inserirInicio(u);
```

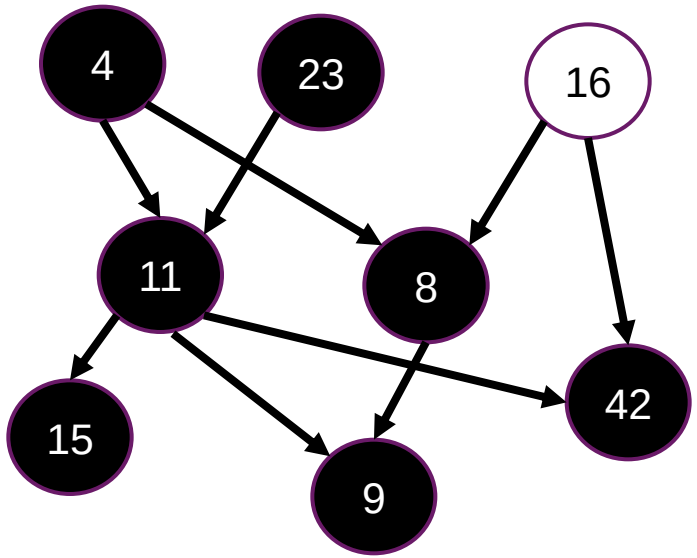


L

4	8	11	42	9	15		

Algoritmo DFS – principal (visita)

```
u.cor = cinza;
Para cada vértice v vizinho de u faça
        se v.cor == branco
                Visitar(v);
        Fim se
Fim Para
u.cor = preto;
L.inserirInicio(u);
```

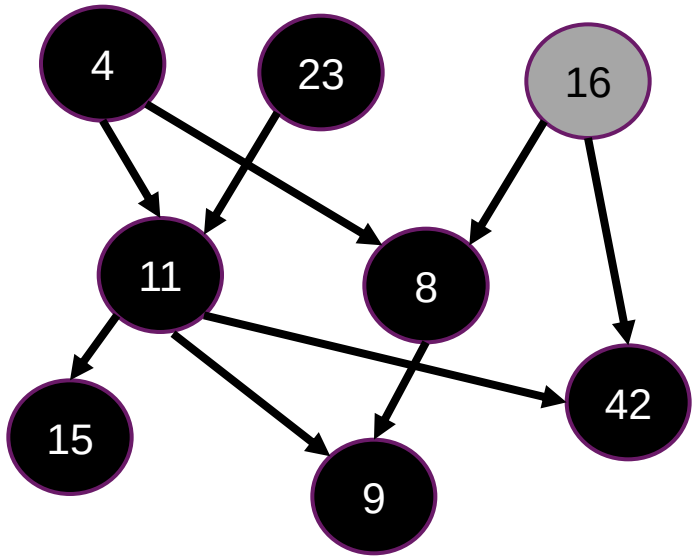


L

23	4	8	11	42	9	15	

Algoritmo DFS – principal (visita)

```
u.cor = cinza;
Para cada vértice v vizinho de u faça
        se v.cor == branco
                Visitar(v);
        Fim se
Fim Para
u.cor = preto;
L.inserirInicio(u);
```

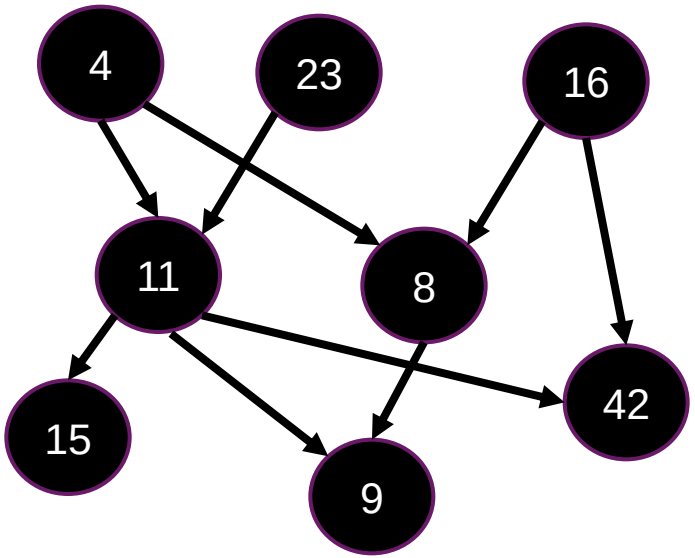


L

23	4	8	11	42	9	15	

Algoritmo DFS – principal (visita)

```
u.cor = cinza;
Para cada vértice v vizinho de u faça
        se v.cor == branco
                Visitar(v);
        Fim se
Fim Para
u.cor = preto;
L.inserirInicio(u);
```



L

16	23	4	8	11	42	9	15

DFS e ordenação

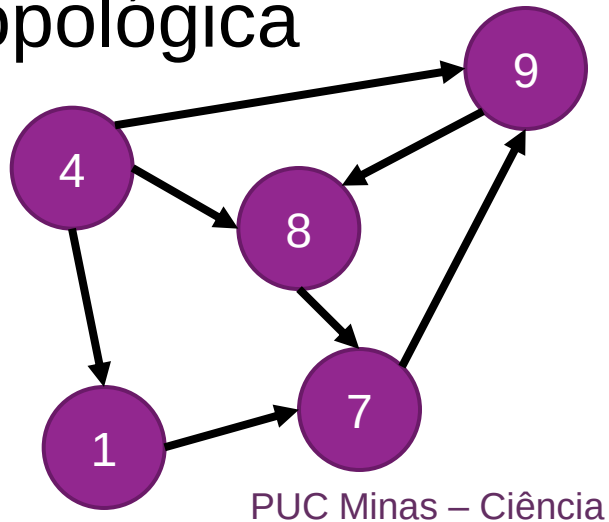
- Como detectar ciclos?
- Relembrando: arestas de retorno
  - Ao alcançarmos um vértice cinza

Ordenação topológica

- Aplicações
  - Planejamento e sequenciamento de tarefas
  - Compilação de módulos
  - Dicionários
  - Pré-requisitos
  - Verificação de dependências (bibliotecas, etc)

Exercícios

- Para o grafo G abaixo, utilize um algoritmo para mostrar um ciclo ou determinar uma ordenação topológica

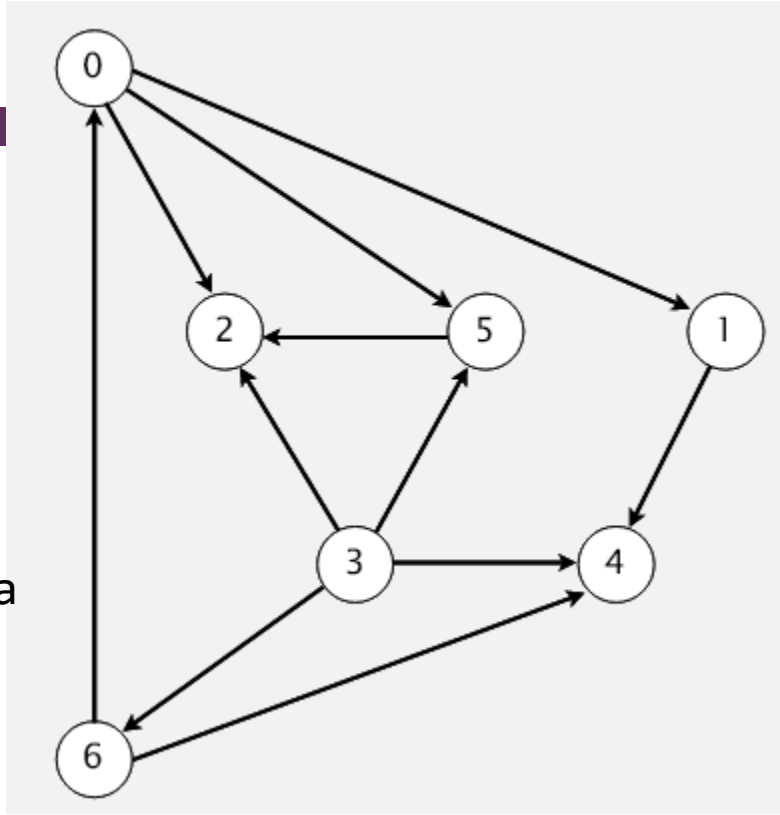



```
u.cor = cinza;
Para cada vértice v vizinho de u faça
        se v.cor == branco

                Visitar(v);
        Fim se
Fim Para
u.cor = preto;
L.inserirInicio(u)
```

Exercícios


```
L = ∅;
S = todos os vértices sem arcos de entrada;
Enquanto S ≠ ∅ faça
   remover um vértice v de S
   inserir o vértice v em L
   para cada arco v,w existente faça
      remover o arco v,w de E
      se w não possuir mais arcos de entrada
          inserir w em S
   Fim para
Fim enquanto
Se E = ∅ retorna L  //lista ordenada
Senão o grafo possui pelo menos um ciclo
```



Exercícios

- Um aluno precisa cursar as disciplinas listadas ao lado, com seus pré-requisitos. Proponha para ele uma ordem na qual as disciplinas podem ser cursadas

Disciplina	Pré-requisitos
AEDs I	---
AEDs II	AEDs I
Matemática Discreta	---
Arquitetura I	Introdução à Computação
Arquitetura II	Arquitetura I
Sistemas Operacionais	Arquitetura II
Grafos	AEDs II
Iniciação à Pesquisa	Introdução à Computação e AEDs II
Introdução à Computação	---
Compiladores	AEDs II e Teoria de Linguagens
Teoria de Linguagens	Matemática Discreta

Exercícios

Disciplina	Pré-requisitos
AEDs I	---
AEDs II	AEDs I
Matemática Discreta	---
Arquitetura I	Introdução à Computação
Arquitetura II	Arquitetura I
Sistemas Operacionais	Arquitetura II
Grafos	AEDs II
Iniciação à Pesquisa	Introdução à Computação e AEDs II
Introdução à Computação	---
Compiladores	AEDs II e Teoria de Linguagens
Teoria de Linguagens	Matemática Discreta

```
L = ∅;
S = todos os vértices sem arcos de e
Enquanto S ≠ ∅ faça
    remover um vértice v de S
    inserir o vértice v em L
    para cada arco v,w existente faç
       remover o arco v,w de E
       se w não possuir mais arcos d
           inserir w em S
    Fim para
Fim enquanto
Se E = ∅ retorna L  //lista ordenada
Senão o grafo possui pelo menos um ciclo
```

OBRIGADO.

Dúvidas?