*Análise de Algoritmos*

# Parte 5: Algoritmos Elementares de Grafos

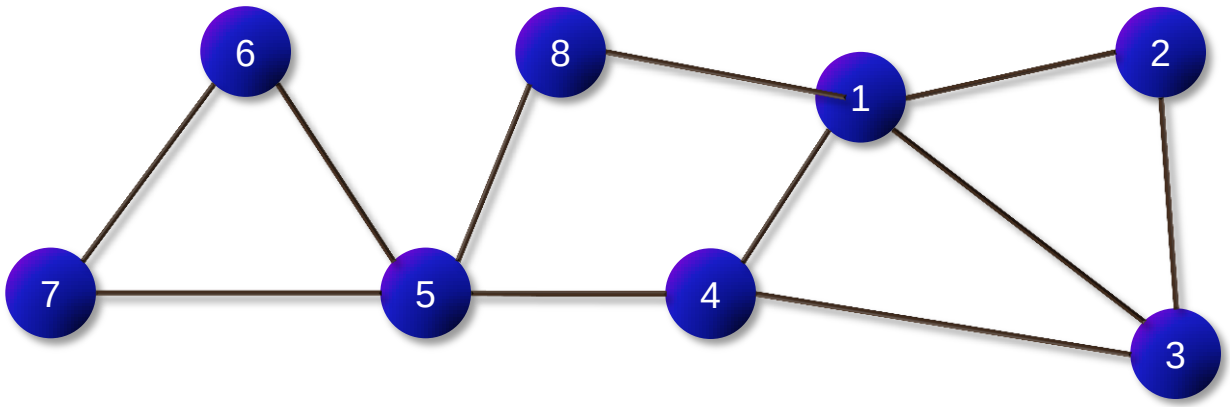
Eduardo Freire Nakamura
eduardo.nakamura@fucapi.br

CESF – FUCAPI
Ciência da Computação, Engenharia da Computação e Sistemas de Informação

# Grafo

- Uma representação gráfica das relações existentes entre elementos de dados

- Ferramenta de modelagem para problemas
  - Conjunto de pontos (vértices) ligados por retas (as arestas)
  - As arestas podem ser direcionadas (setas)
  - Vértices e a restas podem ter pesos (custos) associados



# Grafo

- Úteis na representação de problemas da vida real, em vários campos profissionais

- Exemplos
  - Mapa de estradas: podemos usar algoritmos específicos para determinar o caminho mais curto entre dois pontos

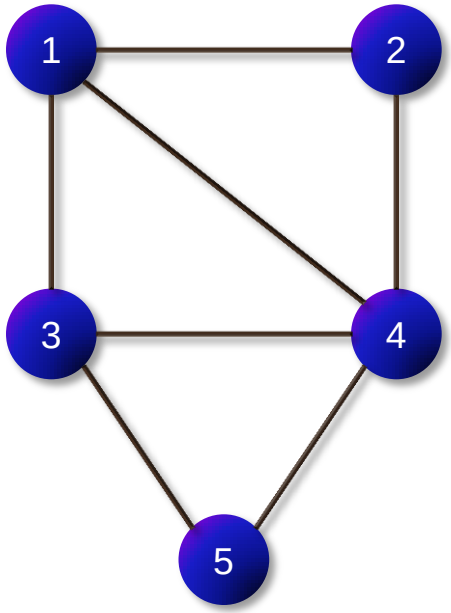
  - Redes de computadores: cada terminal é um vértice, o cabo de rede pelas arestas e o custo associado ao atraso, por exemplo

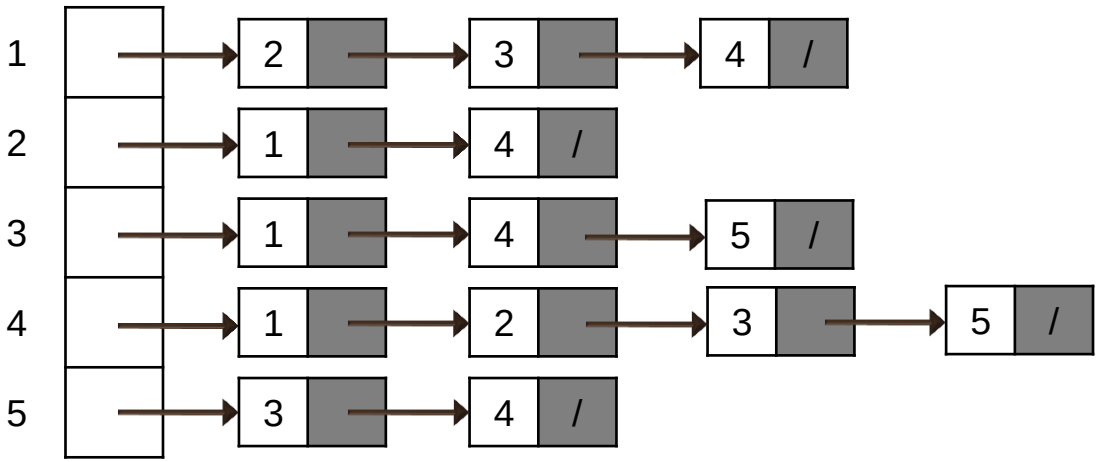
# Representação

- Um grafo $G=(V,E)$ possui
  - V – conjunto de vértices
  - E – conjunto de arestas

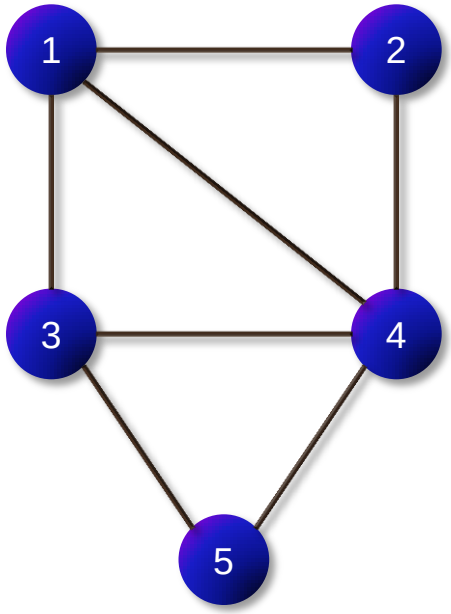
- Duas representações comuns
  - Coleção de listas de adjacências (listas encadeadas)
  - Matriz de adjacência (matriz)

# Listas de adjacência



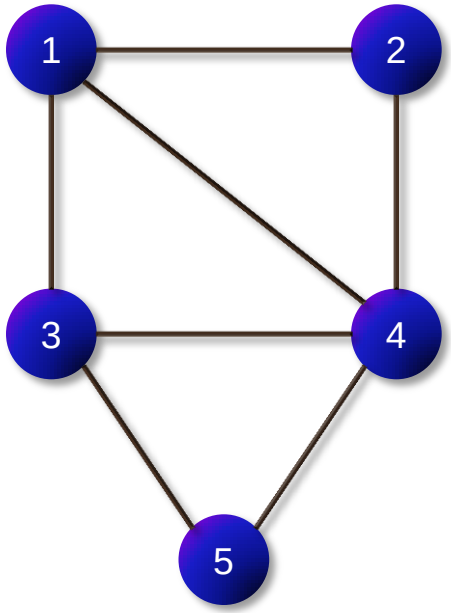



# Matriz de adjacência



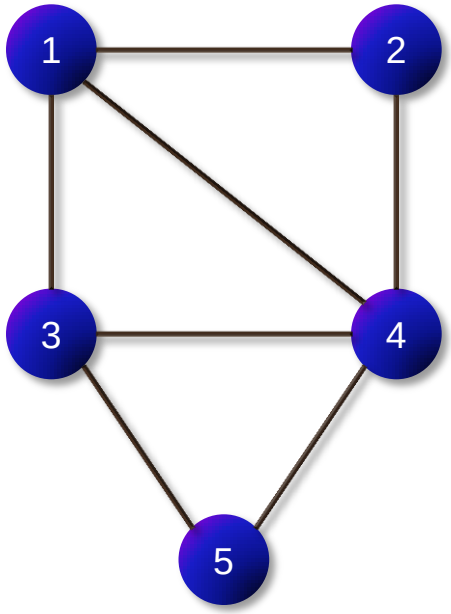

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | | 1 | 1 | 1 | |
| 2 | | | | | |
| 3 | | | | | |
| 4 | | | | | |
| 5 | | | | | |

# Matriz de adjacência



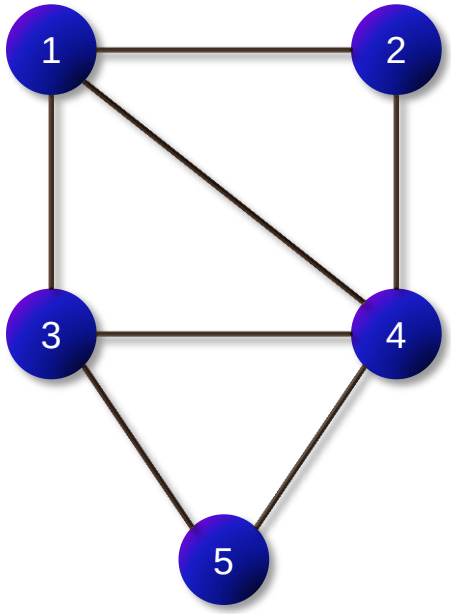

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 1 | 0 |
| 2 | | | | | |
| 3 | | | | | |
| 4 | | | | | |
| 5 | | | | | |

# Matriz de adjacência



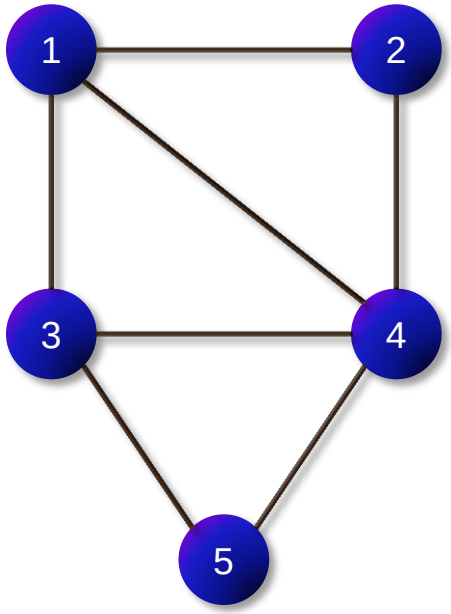

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 1 | 0 |
| 2 | 1 | 0 | 0 | 1 | 0 |
| 3 | | | | | |
| 4 | | | | | |
| 5 | | | | | |

# Matriz de adjacência



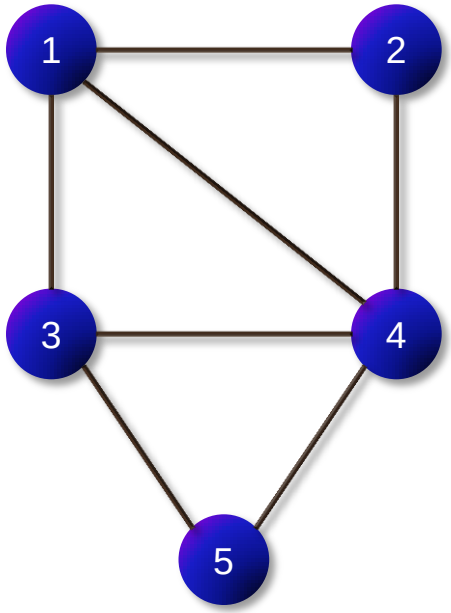

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 1 | 0 |
| 2 | 1 | 0 | 0 | 1 | 0 |
| 3 | 1 | 0 | 0 | 1 | 1 |
| 4 | | | | | |
| 5 | | | | | |

# Matriz de adjacência



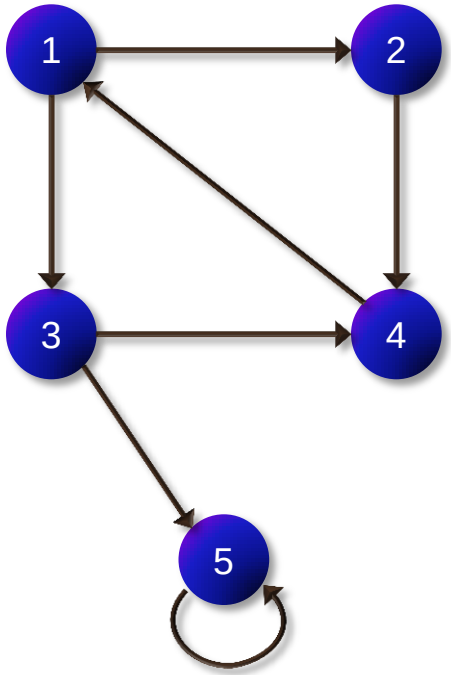

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 1 | 0 |
| 2 | 1 | 0 | 0 | 1 | 0 |
| 3 | 1 | 0 | 0 | 1 | 1 |
| 4 | 1 | 1 | 1 | 0 | 1 |
| 5 | | | | | |

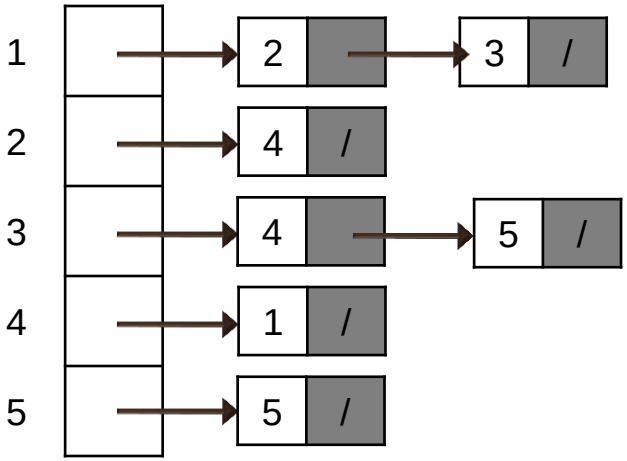
# Matriz de adjacência



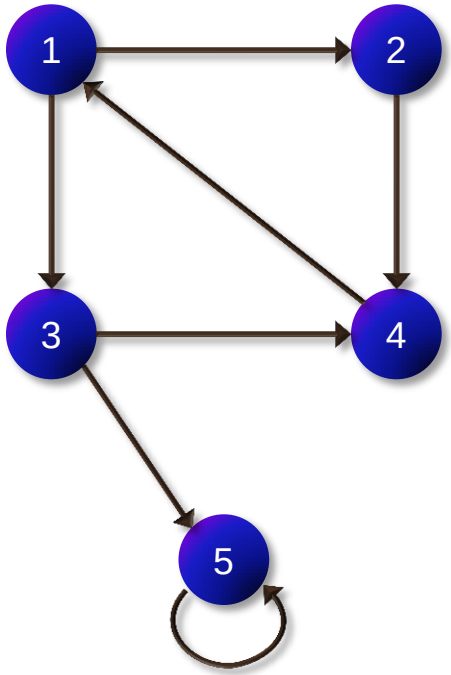

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 1 | 0 |
| 2 | 1 | 0 | 0 | 1 | 0 |
| 3 | 1 | 0 | 0 | 1 | 1 |
| 4 | 1 | 1 | 1 | 0 | 1 |
| 5 | 0 | 0 | 1 | 1 | 0 |

# Para um grafo direcionado (lista)



# Para um grafo direcionado

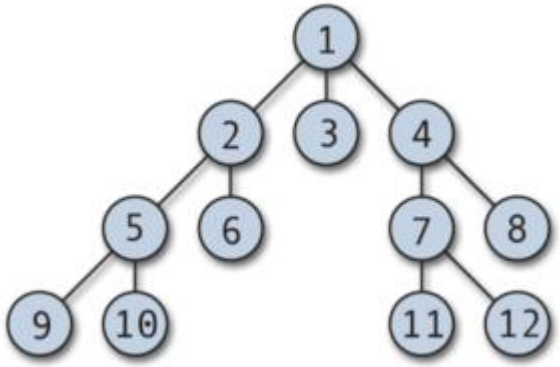



| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 0 | 0 |
| 2 | 0 | 0 | 0 | 1 | 0 |
| 3 | 0 | 0 | 0 | 1 | 1 |
| 4 | 1 | 0 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 | 0 | 1 |

# Busca em largura

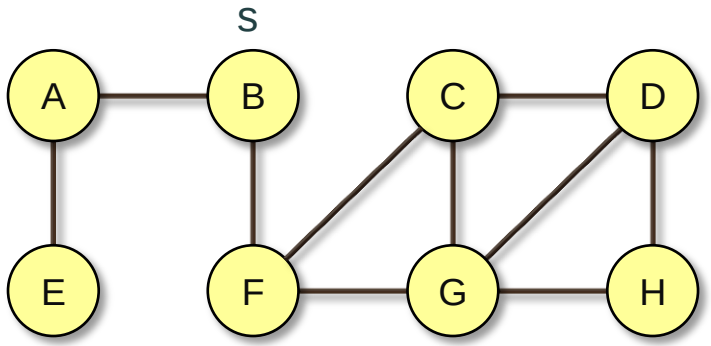
- Algoritmo simples para pesquisa em um grafo
  - Base para muitos algoritmos em grafos

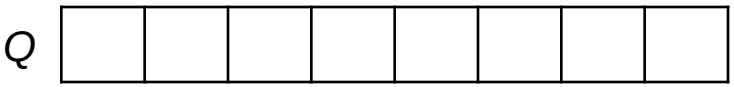
- Dado um grafo $G=(V,E)$ e um vértice de origem $s$
  - O algoritmo visita (pintando) todas as arestas de $G$ até descobrir cada vértice acessível a partir de $s$
  - O algoritmo calcula a distância desde $s$ até todos os vértices alcançáveis

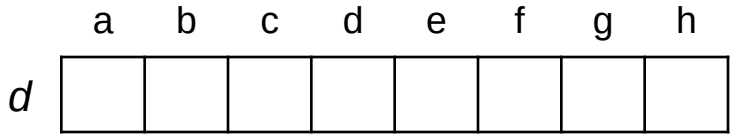


# Busca em largura

```
BFS(G,s)
 1: for each u ∈ (V − {s}) do
 2:   cor[u] ← BRANCO;
 3:   d[u]← ∞;
 4: end for
 5: cor[s] ← CINZA;
 6: d[s] ← 0;
 7: Q ← ∅;
 8: ENQUEUE(Q,s);
 9: while Q ≠ ∅ do
10:   u ← DEQUEUE(Q);
11:   for each v ← Adj[u] do
12:     if cor[v] = BRANCO then
13:       cor[v] ← CINZA;
14:       d[v] ← d[u] + 1;
15:       ENQUEUE(Q,v);
16:     end if
17:   end for
18:   cor[u] ← PRETO;
```

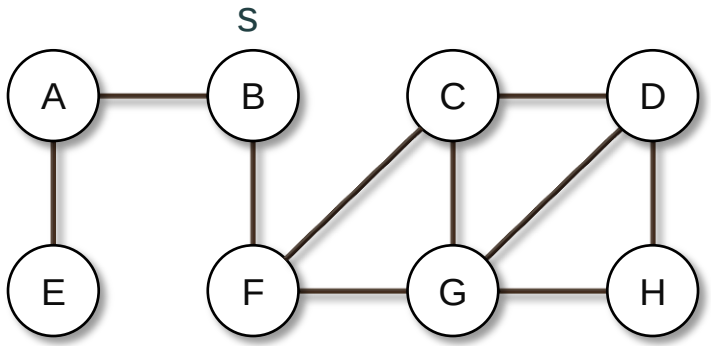


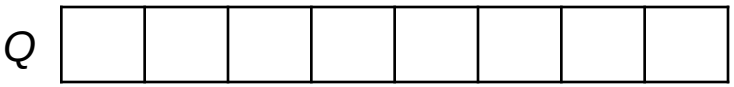



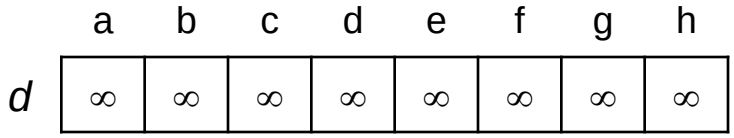



# Busca em largura

```
BFS(G,s)
 1: for each u ∈ (V − {s}) do
 2:   cor[u] ← BRANCO;
 3:   d[u]← ∞;
 4: end for
 5: cor[s] ← CINZA;
 6: d[s] ← 0;
 7: Q ← ∅;
 8: ENQUEUE(Q,s);
 9: while Q ≠ ∅ do
10:   u ← DEQUEUE(Q);
11:   for each v ← Adj[u] do
12:     if cor[v] = BRANCO then
13:       cor[v] ← CINZA;
14:       d[v] ← d[u] + 1;
15:       ENQUEUE(Q,v);
16:     end if
17:   end for
18:   cor[u] ← PRETO;
```

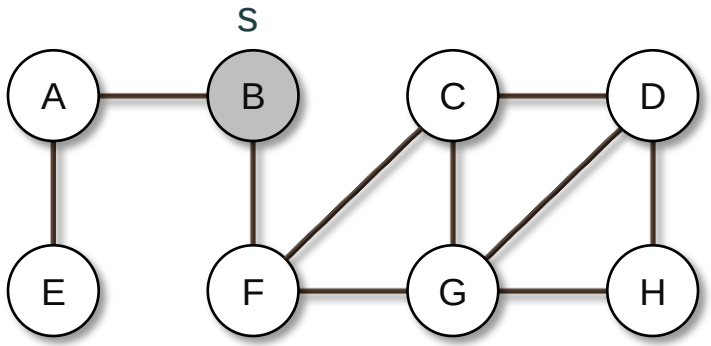


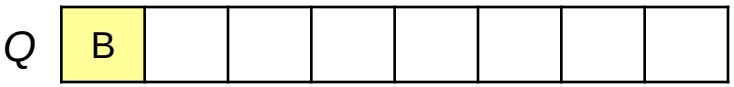



# Busca em largura

```
BFS(G,s)
 1: for each u ∈ (V − {s}) do
 2:   cor[u] ← BRANCO;
 3:   d[u]← ∞;
 4: end for
 5: cor[s] ← CINZA;
 6: d[s] ← 0;
 7: Q ← ∅;
 8: ENQUEUE(Q,s);
 9: while Q ≠ ∅ do
10:   u ← DEQUEUE(Q);
11:   for each v ← Adj[u] do
12:     if cor[v] = BRANCO then
13:       cor[v] ← CINZA;
14:       d[v] ← d[u] + 1;
15:       ENQUEUE(Q,v);
16:     end if
17:   end for
18:   cor[u] ← PRETO;
```

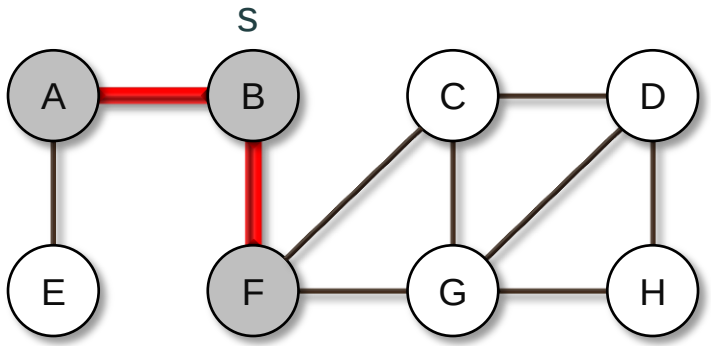



| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| *d* | ∞ | 0 | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ |

# Busca em largura

```
BFS(G,s)
 1: for each u ∈ (V − {s}) do
 2:   cor[u] ← BRANCO;
 3:   d[u]← ∞;
 4: end for
 5: cor[s] ← CINZA;
 6: d[s] ← 0;
 7: Q ← ∅;
 8: ENQUEUE(Q,s);
 9: while Q ≠ ∅ do
10:   u ← DEQUEUE(Q);
11:   for each v ← Adj[u] do
12:     if cor[v] = BRANCO then
13:       cor[v] ← CINZA;
14:       d[v] ← d[u] + 1;
15:       ENQUEUE(Q,v);
16:     end if
17:   end for
18:   cor[u] ← PRETO;
```



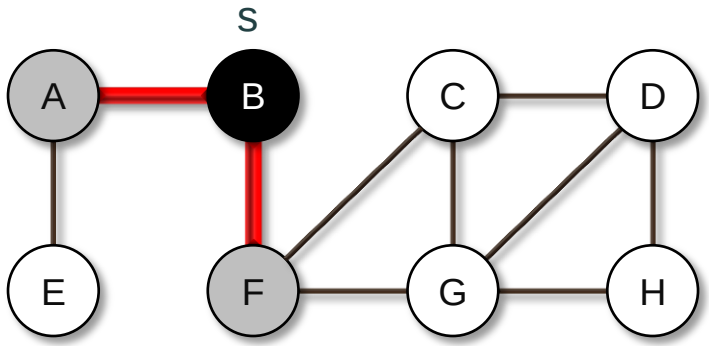

u = B

| Q | F | A | | | | | | |
|---|---|---|---|---|---|---|---|---|

| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| *d* | 1 | 0 | ∞ | ∞ | ∞ | 1 | ∞ | ∞ |

# Busca em largura

```
BFS(G,s)
 1: for each u ∈ (V − {s}) do
 2:   cor[u] ← BRANCO;
 3:   d[u]← ∞;
 4: end for
 5: cor[s] ← CINZA;
 6: d[s] ← 0;
 7: Q ← ∅;
 8: ENQUEUE(Q,s);
 9: while Q ≠ ∅ do
10:   u ← DEQUEUE(Q);
11:   for each v ← Adj[u] do
12:     if cor[v] = BRANCO then
13:       cor[v] ← CINZA;
14:       d[v] ← d[u] + 1;
15:       ENQUEUE(Q,v);
16:     end if
17:   end for
18:   cor[u] ← PRETO;
```




*u* = B

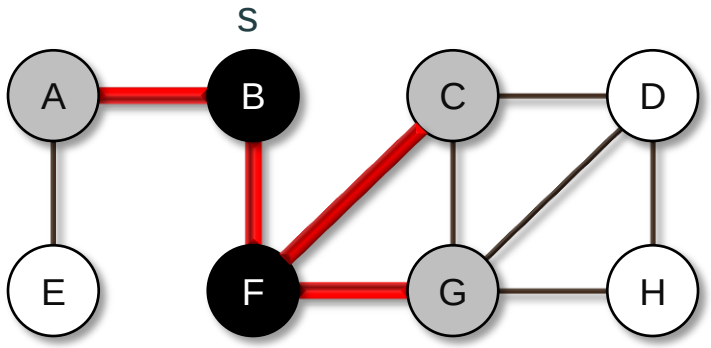
| *Q* | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | F | A | | | | | | |

| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| *d* | 1 | 0 | ∞ | ∞ | ∞ | 1 | ∞ | ∞ |

*d*[*u*] = 0

# Busca em largura

```
BFS(G,s)
 1: for each u ∈ (V − {s}) do
 2:   cor[u] ← BRANCO;
 3:   d[u]← ∞;
 4: end for
 5: cor[s] ← CINZA;
 6: d[s] ← 0;
 7: Q ← ∅;
 8: ENQUEUE(Q,s);
 9: while Q ≠ ∅ do
10:   u ← DEQUEUE(Q);
11:   for each v ← Adj[u] do
12:     if cor[v] = BRANCO then
13:       cor[v] ← CINZA;
14:       d[v] ← d[u] + 1;
15:       ENQUEUE(Q,v);
16:     end if
17:   end for
18:   cor[u] ← PRETO;
```




*u* = F

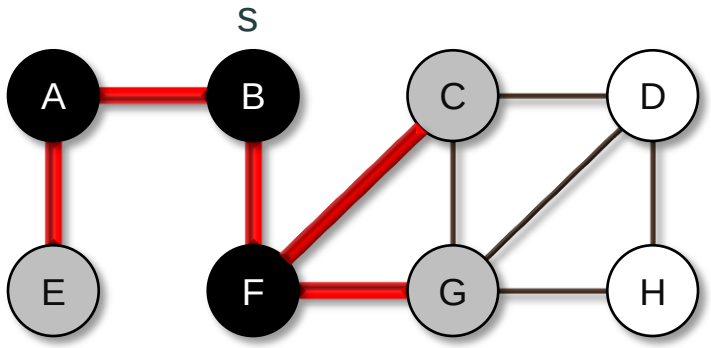
| *Q* | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | A | C | G | | | | | |

| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| *d* | 1 | 0 | 2 | ∞ | ∞ | 1 | 2 | ∞ |

*d*[*u*] = 1

# Busca em largura

```
BFS(G,s)
 1: for each u ∈ (V − {s}) do
 2:   cor[u] ← BRANCO;
 3:   d[u]← ∞;
 4: end for
 5: cor[s] ← CINZA;
 6: d[s] ← 0;
 7: Q ← ∅;
 8: ENQUEUE(Q,s);
 9: while Q ≠ ∅ do
10:   u ← DEQUEUE(Q);
11:   for each v ← Adj[u] do
12:     if cor[v] = BRANCO then
13:       cor[v] ← CINZA;
14:       d[v] ← d[u] + 1;
15:       ENQUEUE(Q,v);
16:     end if
17:   end for
18:   cor[u] ← PRETO;
```




*u* = A

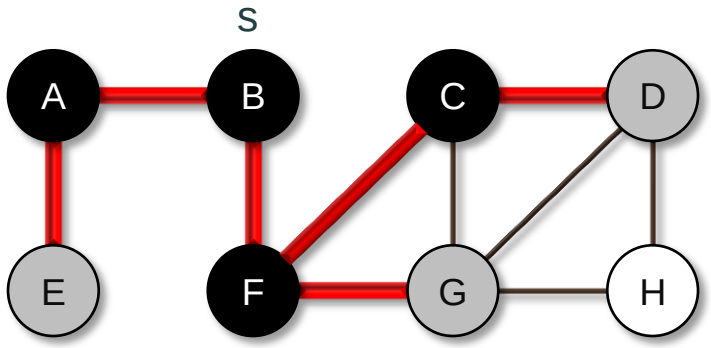
| *Q* | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | C | G | E | | | | | |

| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| *d* | 1 | 0 | 2 | ∞ | 2 | 1 | 2 | ∞ |

*d*[*u*] = 1

# Busca em largura

```
BFS(G,s)
 1: for each u ∈ (V − {s}) do
 2:   cor[u] ← BRANCO;
 3:   d[u]← ∞;
 4: end for
 5: cor[s] ← CINZA;
 6: d[s] ← 0;
 7: Q ← ∅;
 8: ENQUEUE(Q,s);
 9: while Q ≠ ∅ do
10:   u ← DEQUEUE(Q);
11:   for each v ← Adj[u] do
12:     if cor[v] = BRANCO then
13:       cor[v] ← CINZA;
14:       d[v] ← d[u] + 1;
15:       ENQUEUE(Q,v);
16:     end if
17:   end for
18:   cor[u] ← PRETO;
```




*u* = C

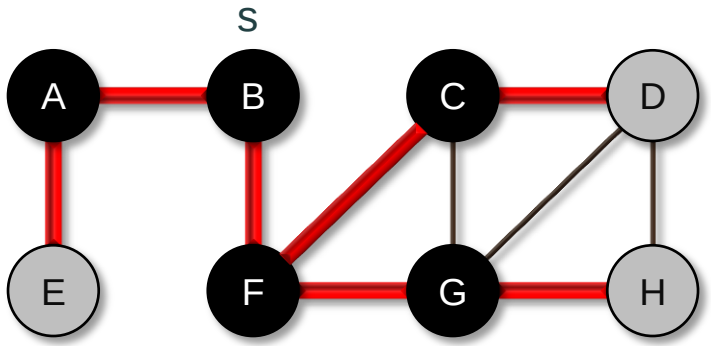
| *Q* | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | G | E | D | | | | | |

| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| *d* | 1 | 0 | 2 | 3 | 2 | 1 | 2 | ∞ |

*d*[*u*] = 2

# Busca em largura

```
BFS(G,s)
 1: for each u ∈ (V − {s}) do
 2:   cor[u] ← BRANCO;
 3:   d[u]← ∞;
 4: end for
 5: cor[s] ← CINZA;
 6: d[s] ← 0;
 7: Q ← ∅;
 8: ENQUEUE(Q,s);
 9: while Q ≠ ∅ do
10:   u ← DEQUEUE(Q);
11:   for each v ← Adj[u] do
12:     if cor[v] = BRANCO then
13:       cor[v] ← CINZA;
14:       d[v] ← d[u] + 1;
15:       ENQUEUE(Q,v);
16:     end if
17:   end for
18:   cor[u] ← PRETO;
```




*u* = G

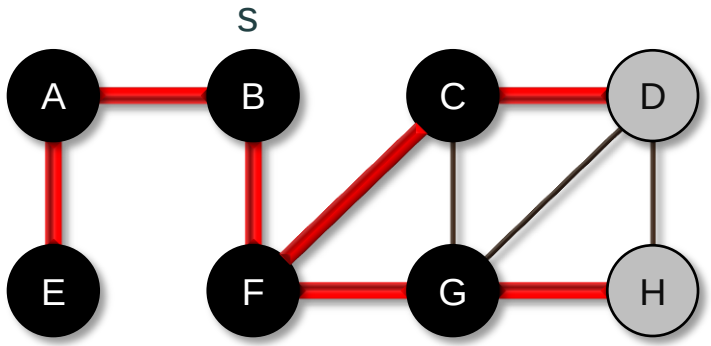
| *Q* | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | E | D | H | | | | | |

| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| *d* | 1 | 0 | 2 | 3 | 2 | 1 | 2 | 3 |

*d*[*u*] = 2

# Busca em largura

```
BFS(G,s)
 1: for each u ∈ (V − {s}) do
 2:   cor[u] ← BRANCO;
 3:   d[u]← ∞;
 4: end for
 5: cor[s] ← CINZA;
 6: d[s] ← 0;
 7: Q ← ∅;
 8: ENQUEUE(Q,s);
 9: while Q ≠ ∅ do
10:   u ← DEQUEUE(Q);
11:   for each v ← Adj[u] do
12:     if cor[v] = BRANCO then
13:       cor[v] ← CINZA;
14:       d[v] ← d[u] + 1;
15:       ENQUEUE(Q,v);
16:     end if
17:   end for
18:   cor[u] ← PRETO;
```




*u* = E

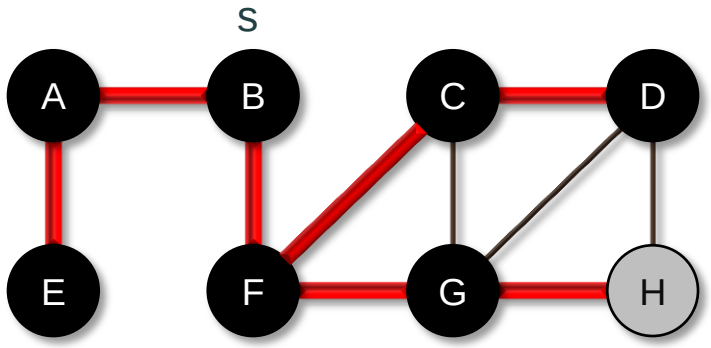
| *Q* | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | D | H | | | | | | |

| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| *d* | 1 | 0 | 2 | 3 | 2 | 1 | 2 | 3 |

*d*[*u*] = 2

# Busca em largura

```
BFS(G,s)
 1: for each u ∈ (V − {s}) do
 2:   cor[u] ← BRANCO;
 3:   d[u]← ∞;
 4: end for
 5: cor[s] ← CINZA;
 6: d[s] ← 0;
 7: Q ← ∅;
 8: ENQUEUE(Q,s);
 9: while Q ≠ ∅ do
10:   u ← DEQUEUE(Q);
11:   for each v ← Adj[u] do
12:     if cor[v] = BRANCO then
13:       cor[v] ← CINZA;
14:       d[v] ← d[u] + 1;
15:       ENQUEUE(Q,v);
16:     end if
17:   end for
18:   cor[u] ← PRETO;
```




*u* = D

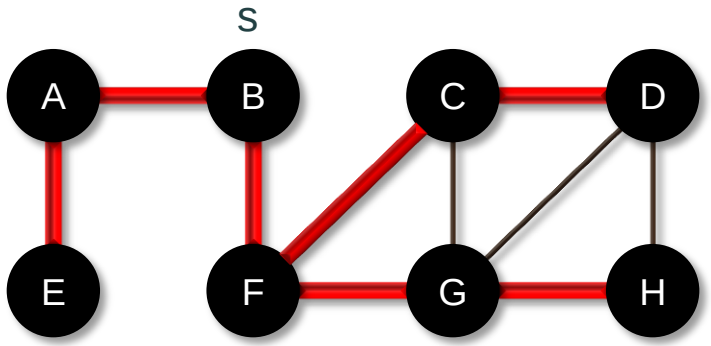
| *Q* | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | H | | | | | | | |

| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| *d* | 1 | 0 | 2 | 3 | 2 | 1 | 2 | 3 |

*d*[*u*] = 3

# Busca em largura

```
BFS(G,s)
 1: for each u ∈ (V − {s}) do
 2:   cor[u] ← BRANCO;
 3:   d[u]← ∞;
 4: end for
 5: cor[s] ← CINZA;
 6: d[s] ← 0;
 7: Q ← ∅;
 8: ENQUEUE(Q,s);
 9: while Q ≠ ∅ do
10:   u ← DEQUEUE(Q);
11:   for each v ← Adj[u] do
12:     if cor[v] = BRANCO then
13:       cor[v] ← CINZA;
14:       d[v] ← d[u] + 1;
15:       ENQUEUE(Q,v);
16:     end if
17:   end for
18:   cor[u] ← PRETO;
```




*u* = H

*Q*

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | | | |

| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| *d* | 1 | 0 | 2 | 3 | 2 | 1 | 2 | 3 |

*d*[*u*] = 3

# Análise

```
BFS(G,s)
 1: for each u ∈ (V − {s}) do
 2:   cor[u] ← BRANCO;
 3:   d[u]← ∞;
 4: end for
 5: cor[s] ← CINZA;
 6: d[s] ← 0;
 7: Q ← ∅;
 8: ENQUEUE(Q,s);
 9: while Q ≠ ∅ do
10:   u ← DEQUEUE(Q);
11:   for each v ← Adj[u] do
12:     if cor[v] = BRANCO then
13:       cor[v] ← CINZA;
14:       d[v] ← d[u] + 1;
15:       ENQUEUE(Q,v);
16:     end if
17:   end for
18:   cor[u] ← PRETO;
```

Todos os vértices são visitados três vezes

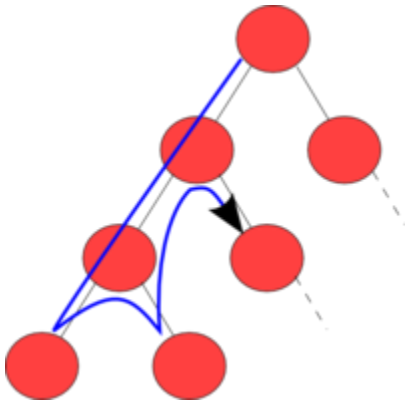
As arestas serão visitadas no máximo uma vez

O custo é $O(V + E)$

# Busca em profundidade

- Algoritmo simples para pesquisa em um grafo
  - Base para muitos algoritmos em grafos
  - Procura o mais distante (profundo) primeiro

- Dado um grafo $G=(V,E)$
  - O algoritmo marca dois carimbos de tempo em cada vértice *u*
    - $d[u]$ –momento em que *u* é descoberto a primeira vez
    - $f[u]$ – momento em que *u* termina de ser examinado

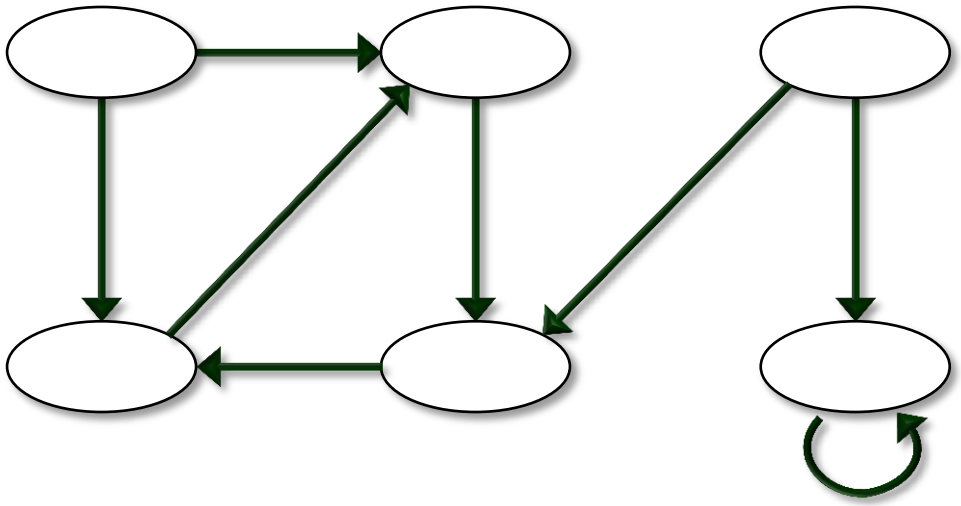


# Busca em profundidade

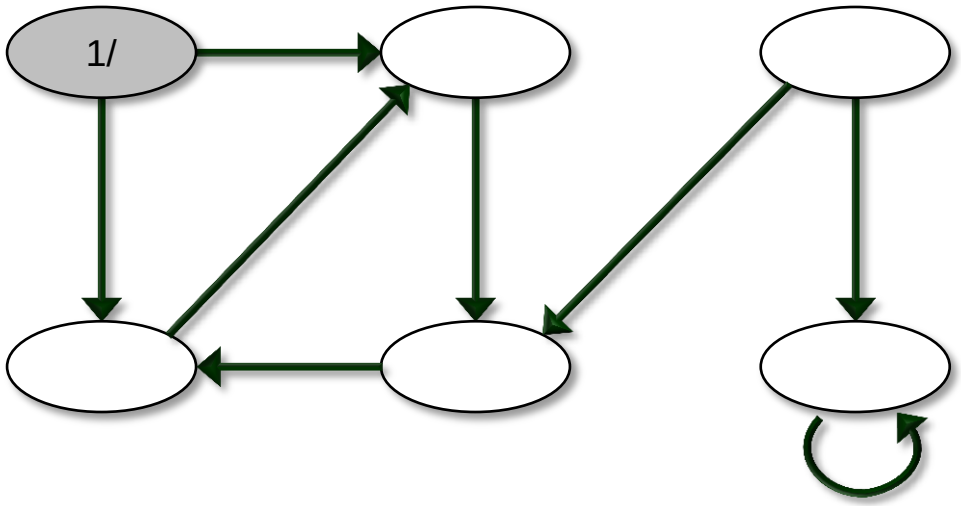
```
DFS(G)
 1: for each u ∈ V do
 2:   cor[u] ← BRANCO;
 3: end for
 4: t ← 0;
 5: for each u ∈ V do
 6:   if cor[u] = BRANCO then
 7:     DFS-VISIT(u);
 8:   end if
 9: end for
```

```
DFS-VISIT(u)
 1: cor[u] ← CINZA;
 2: t ← t + 1;
 3: d[u] ← t;
 4: for each v ∈ Adj[u] then
 5:   if cor[u] = BRANCO then
 6:     DFS-VISIT(v);
 7:   end if
 8: end for
 9: cor[u] ← PRETO;
10: t ← t + 1;
11: f[u] ← t;
```

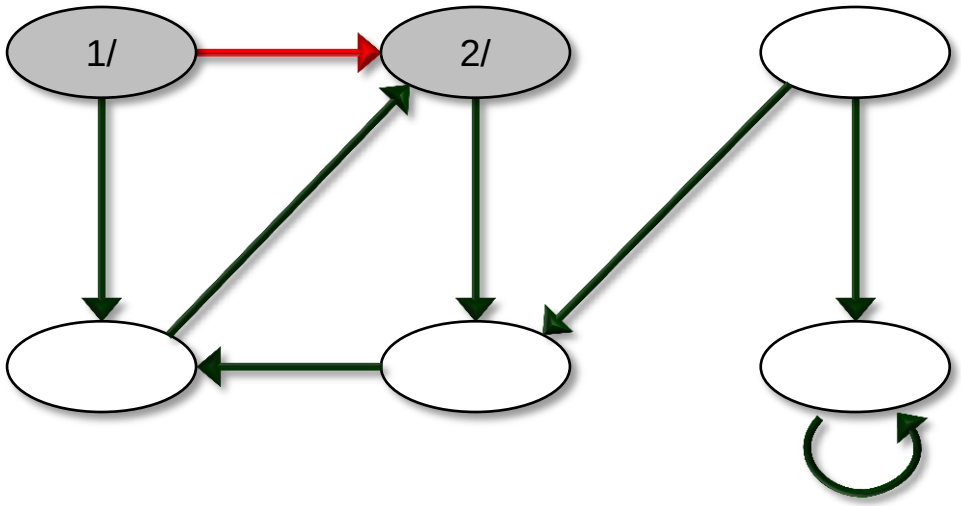
# Busca em profundidade



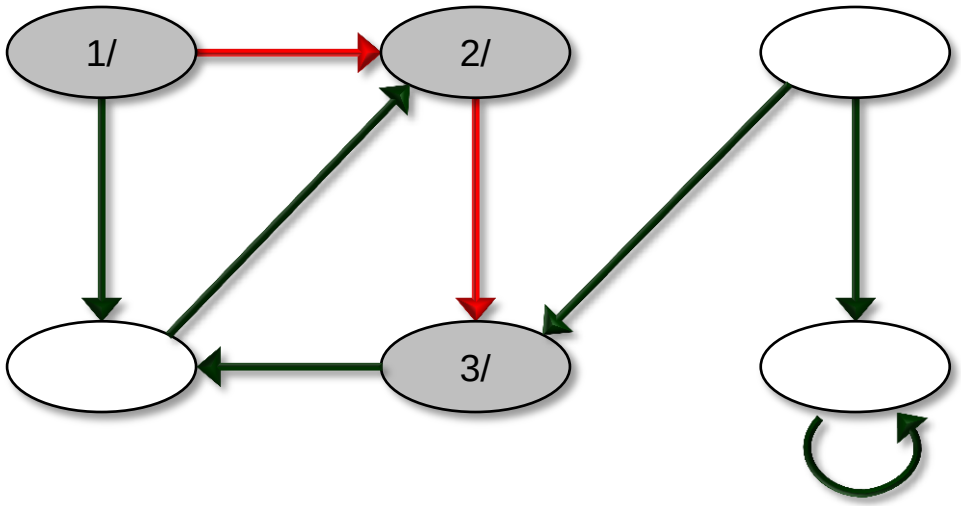
# Busca em profundidade



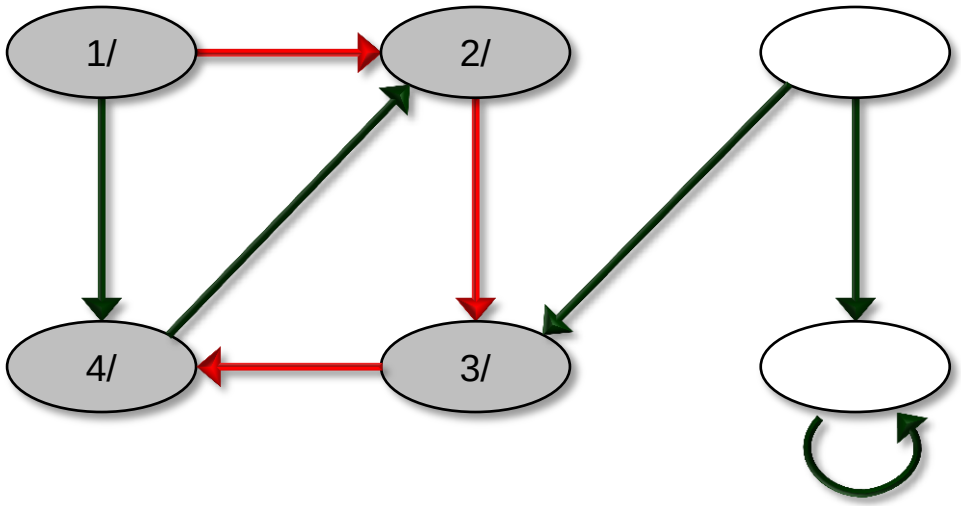
# Busca em profundidade



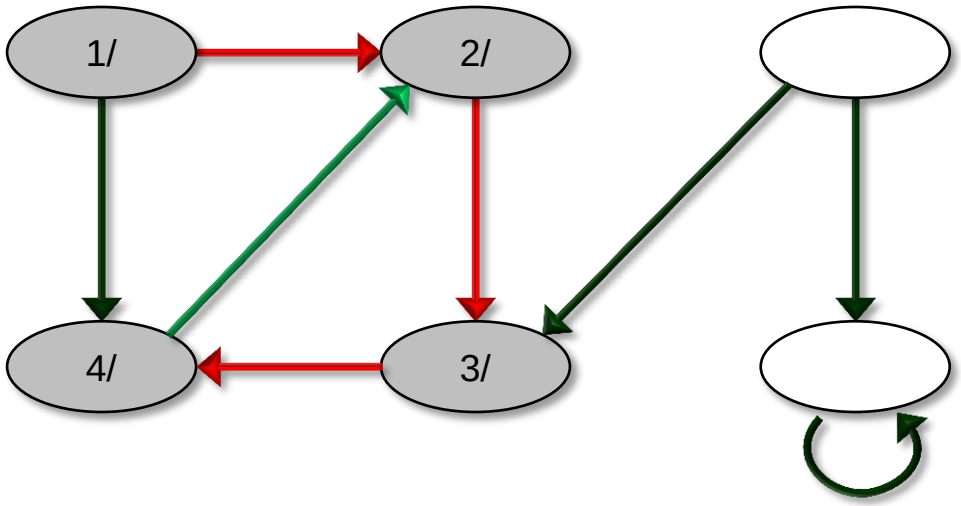
# Busca em profundidade



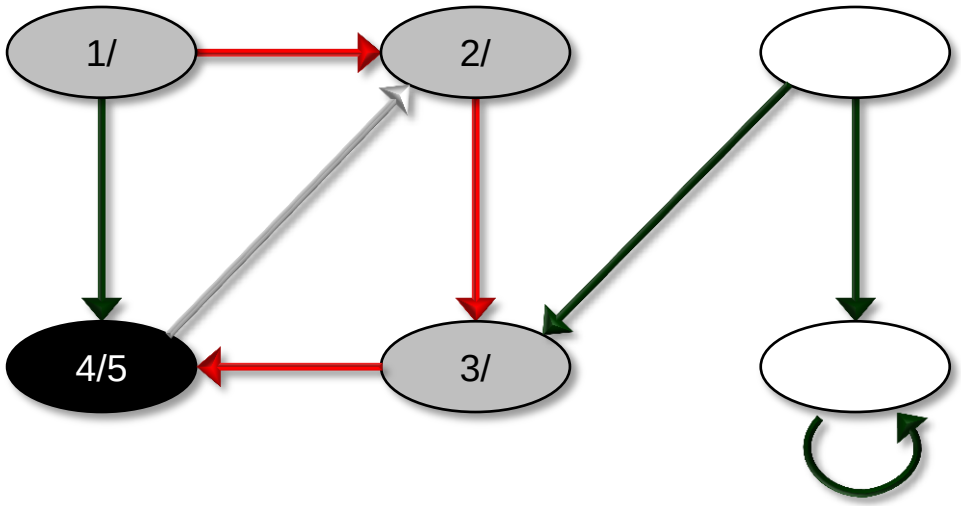
# Busca em profundidade



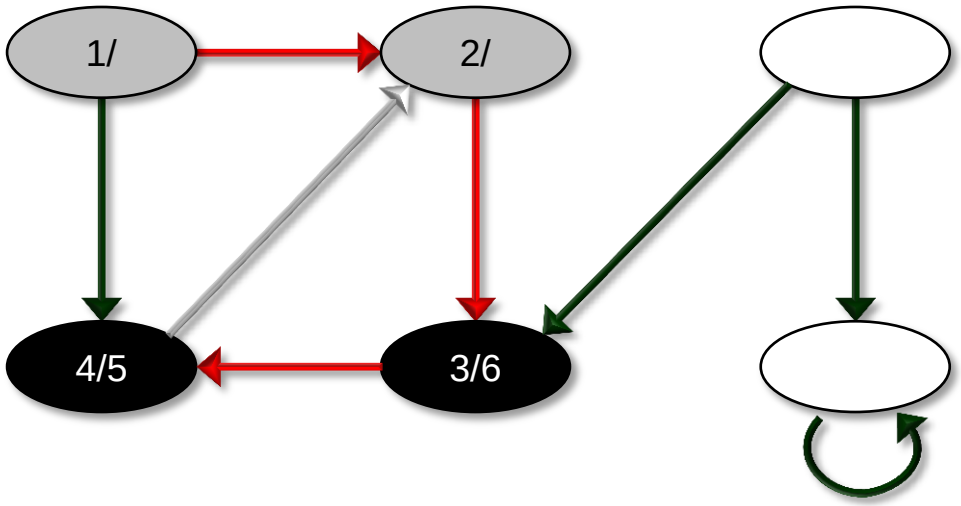
# Busca em profundidade



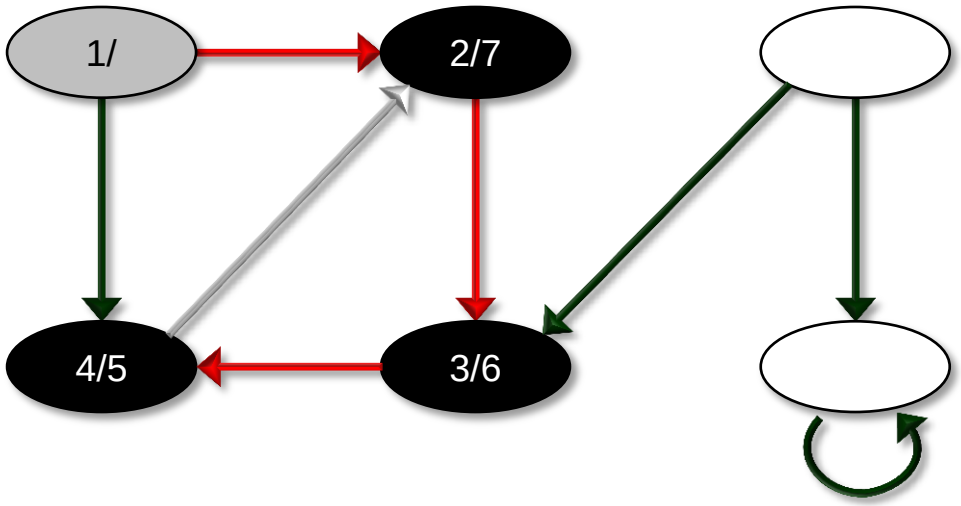
# Busca em profundidade



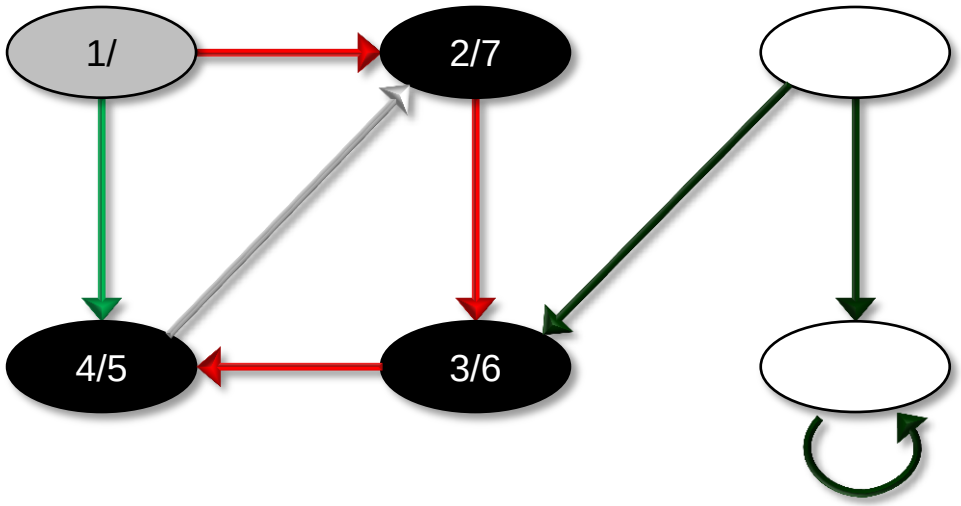
# Busca em profundidade



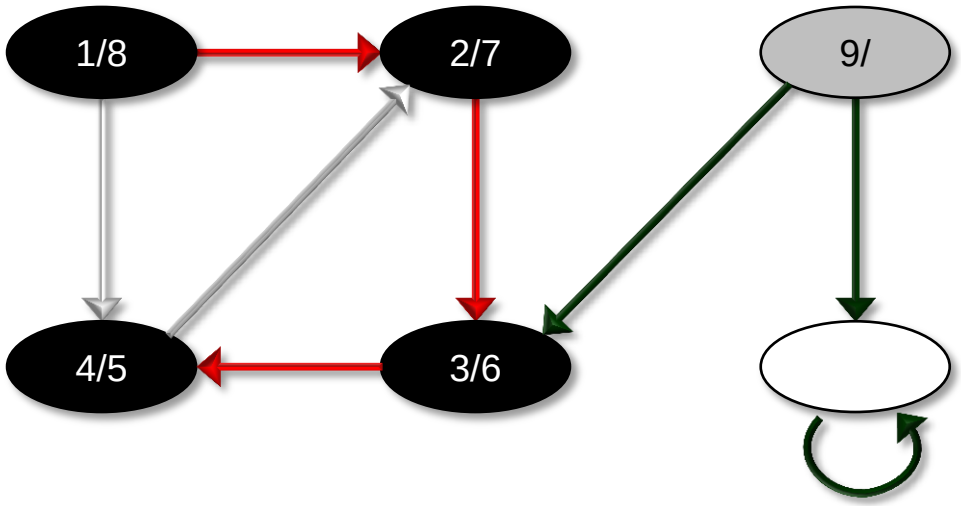
# Busca em profundidade



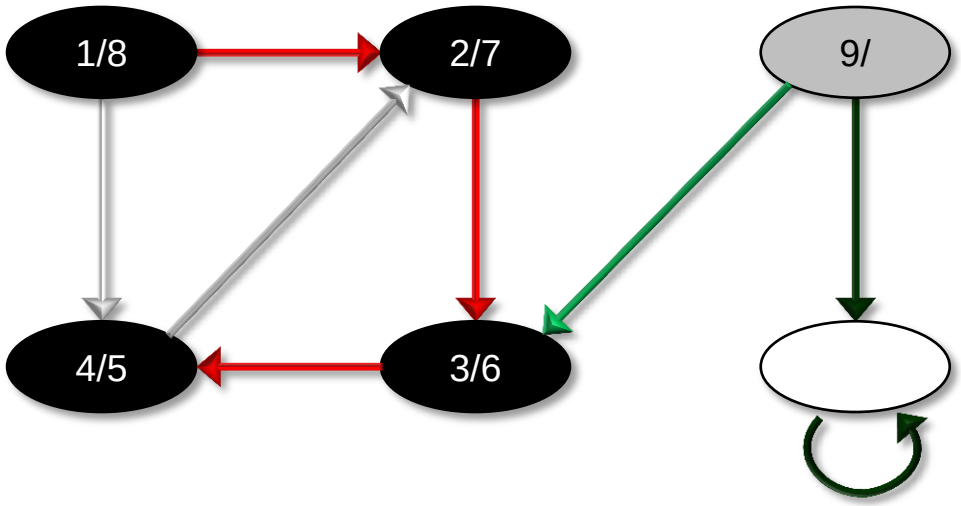
# Busca em profundidade



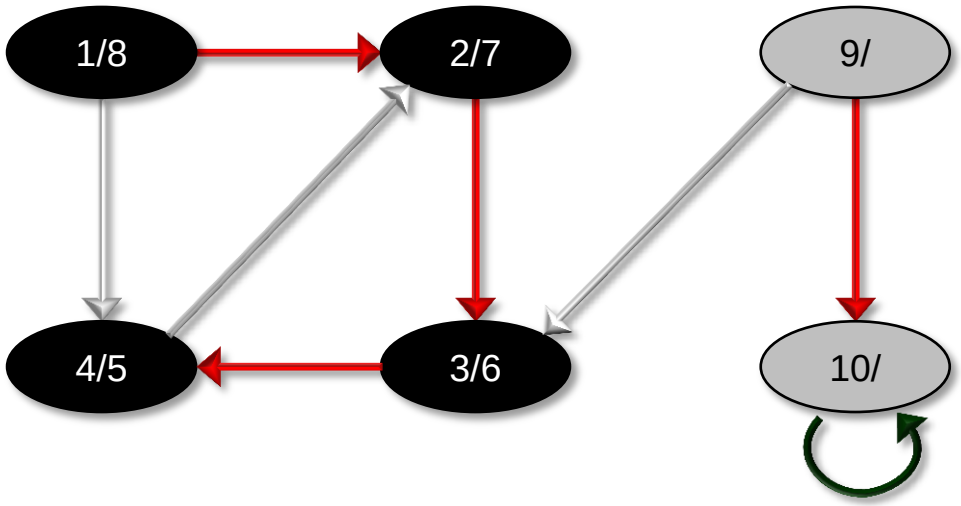
# Busca em profundidade



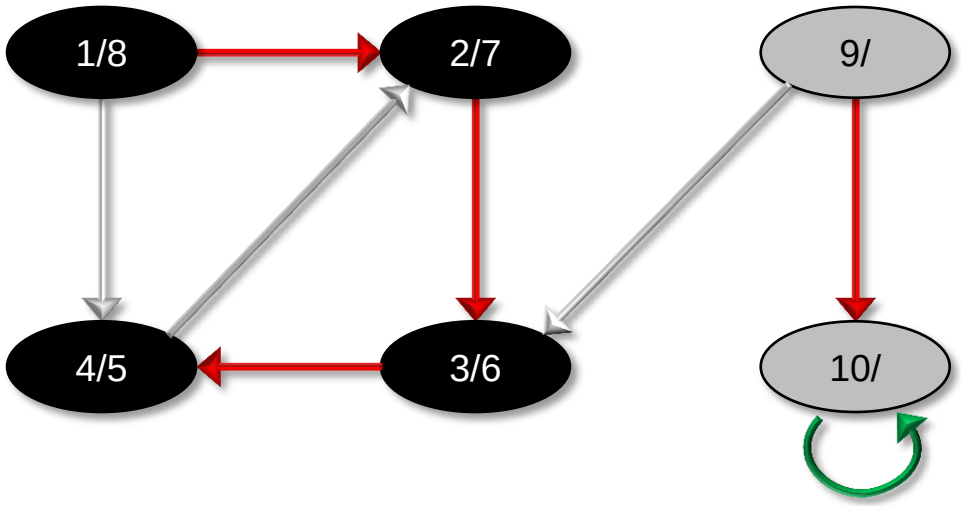
# Busca em profundidade



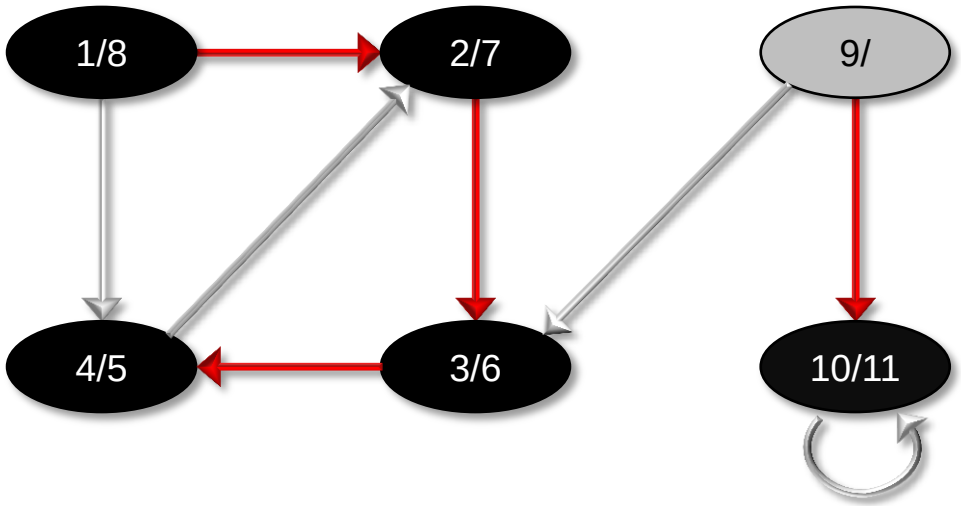
# Busca em profundidade



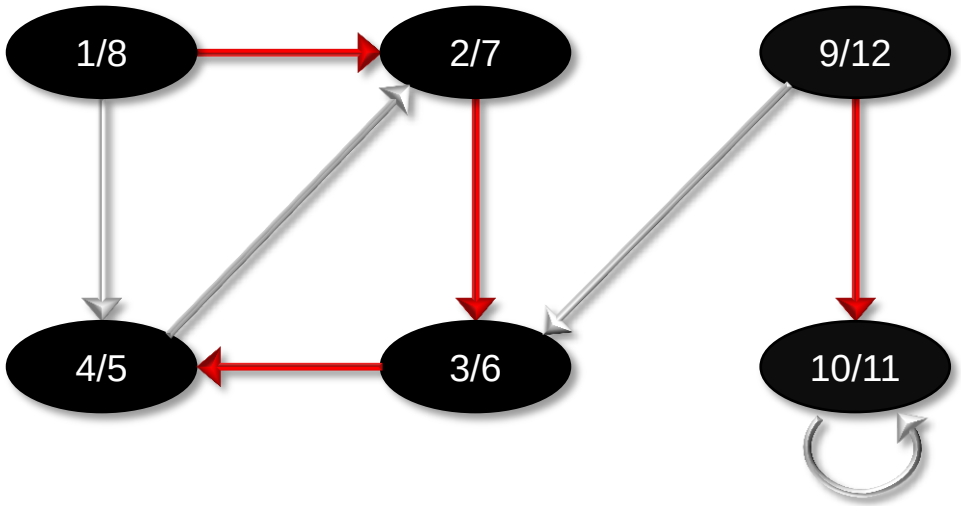
# Busca em profundidade



# Busca em profundidade



# Busca em profundidade



# Análise

Qual é o custo?

# Ordenação topológica

- Ordenação de eventos (dependências)
  - Grafos acíclicos direcionados (orientados), chamados "gaos"
- Ordenação linear dos vértices
  - Se existe uma aresta (*u*,*v*), então *u* aparece antes de *v*
  - Se o grafo possui ciclos não é possível!

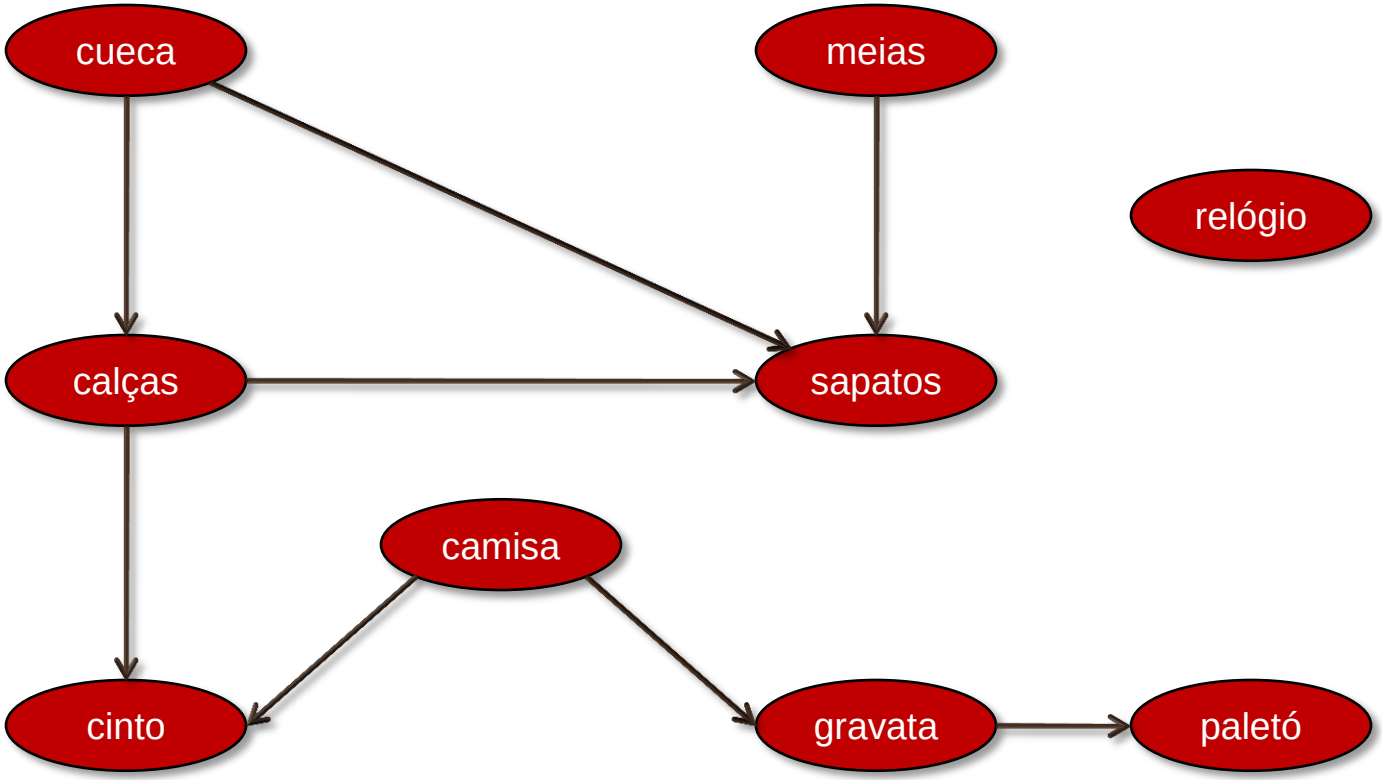
# Ordenação topológica

```
TOPOLOGICAL-SORT(G)
 1: DFS(G);
 2: A medida que f[u] é calculado, inserir
    o vértice u na frente de uma lista list;
 3: return list;
```

# Exemplo

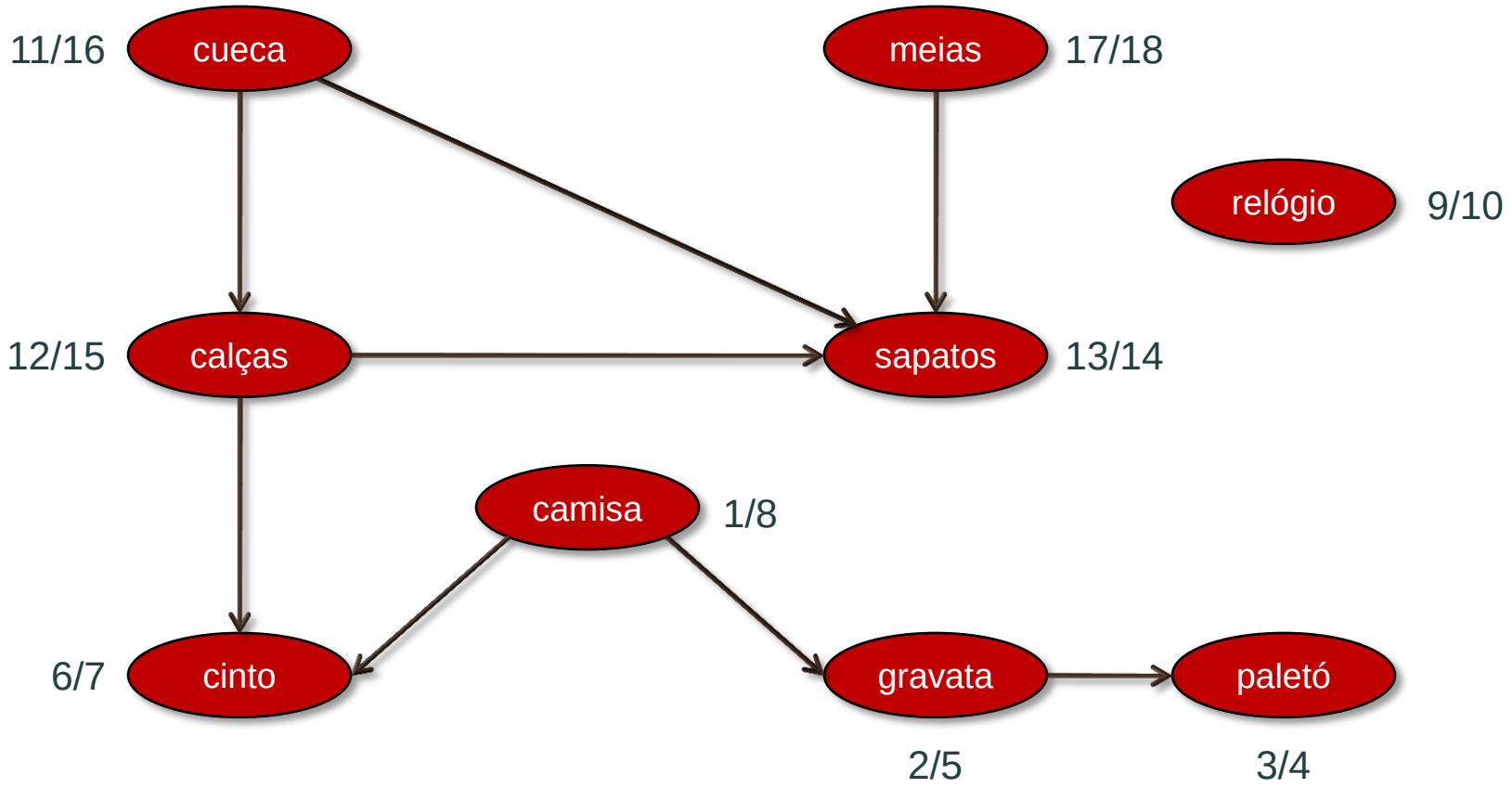
- O professor Manuel Joaquim quer ordenar topologicamente a sua roupa ao se vestir
- As dependências são explicitadas pelo grafo a seguir

# Exemplo

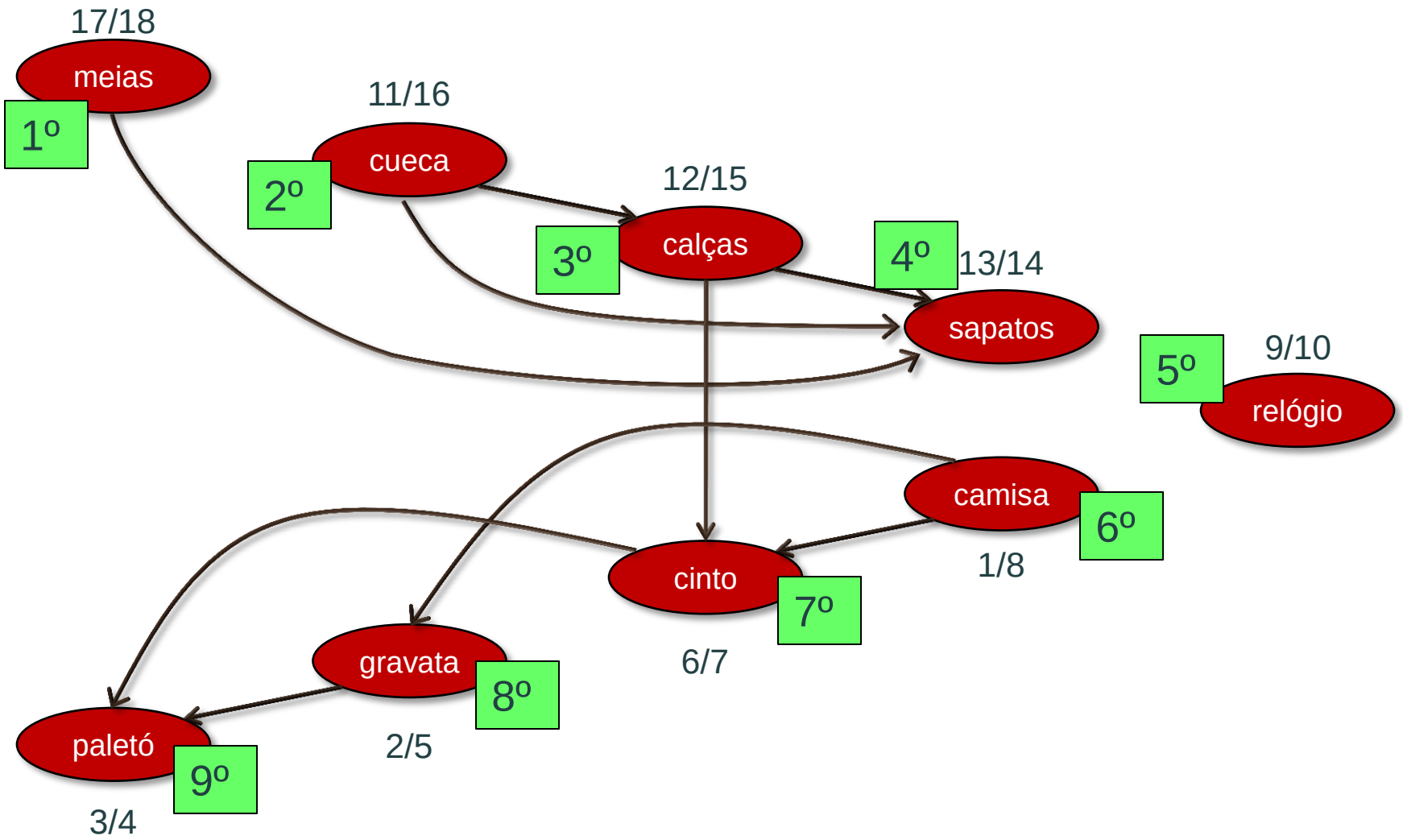

cueca
meias
relógio
calças
sapatos
camisa
cinto
gravata
paletó

# Exemplo

## Resultado da Busca em profundidade



# Exemplo


17/18
meias
1º
11/16
cueca
2º
12/15
calças
3º
4º
13/14
sapatos
5º
9/10
relógio
camisa
6º
1/8
cinto
7º
6/7
gravata
8º
2/5
paletó
9º
3/4

# Exercício

- Dado um grafo orientado representado por uma lista de adjacências
  - Qual é o tempo para calcular o grau de saída (# de arestas que saem do vértice) de todos os vértices?
  - Qual é o tempo para calcular o grau de entrada (# de arestas que chegam no vértice) de todos os vértices?

- Adapte a busca em largura para encontrar o menor caminho entre dois nós *u* e *v*, de um grafo *G*=(*V*,*E*) visitando o menor número de vértices possível