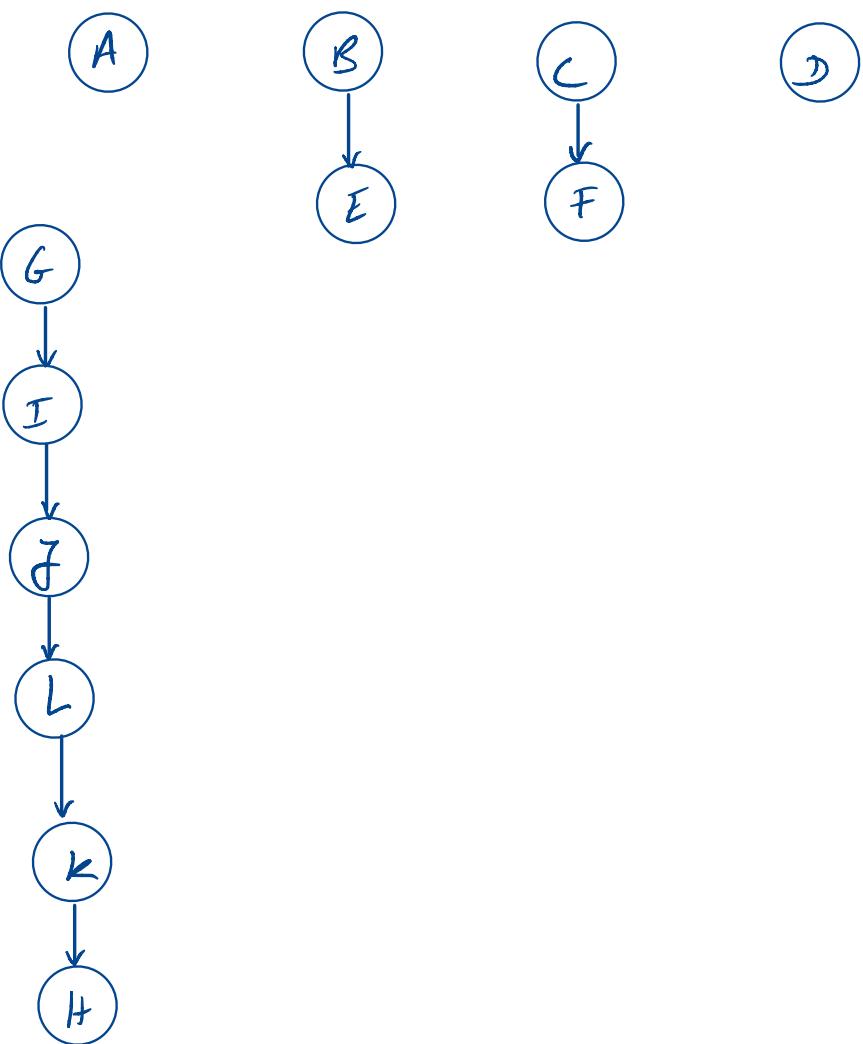
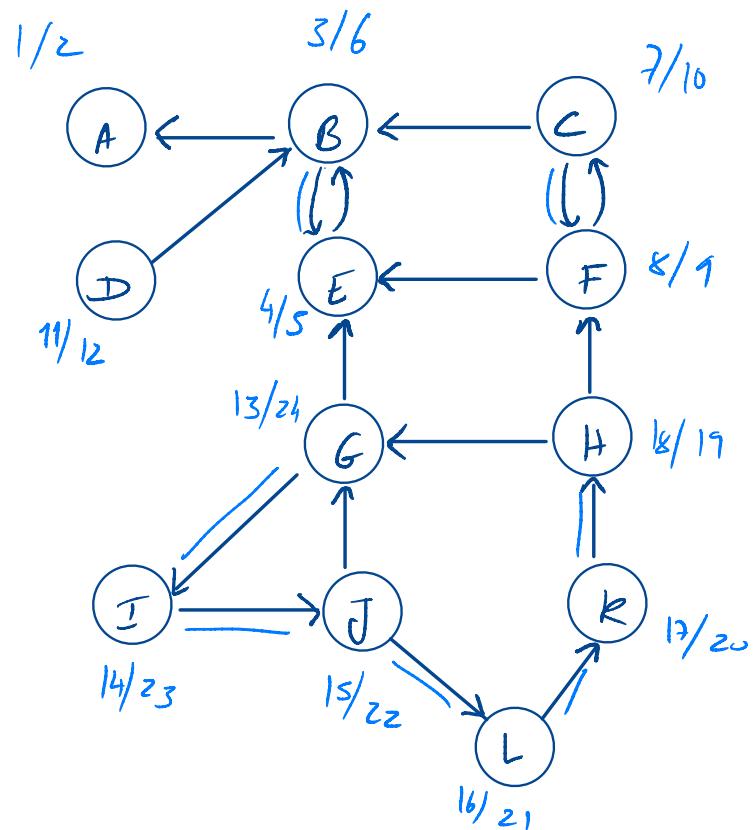


- Componentes Finitamente Ligados
  - Algoritmo de Karpinski-Sharir
  - Algoritmo de Tarjan
- Procura em Profundidade Primeiro (BFS)

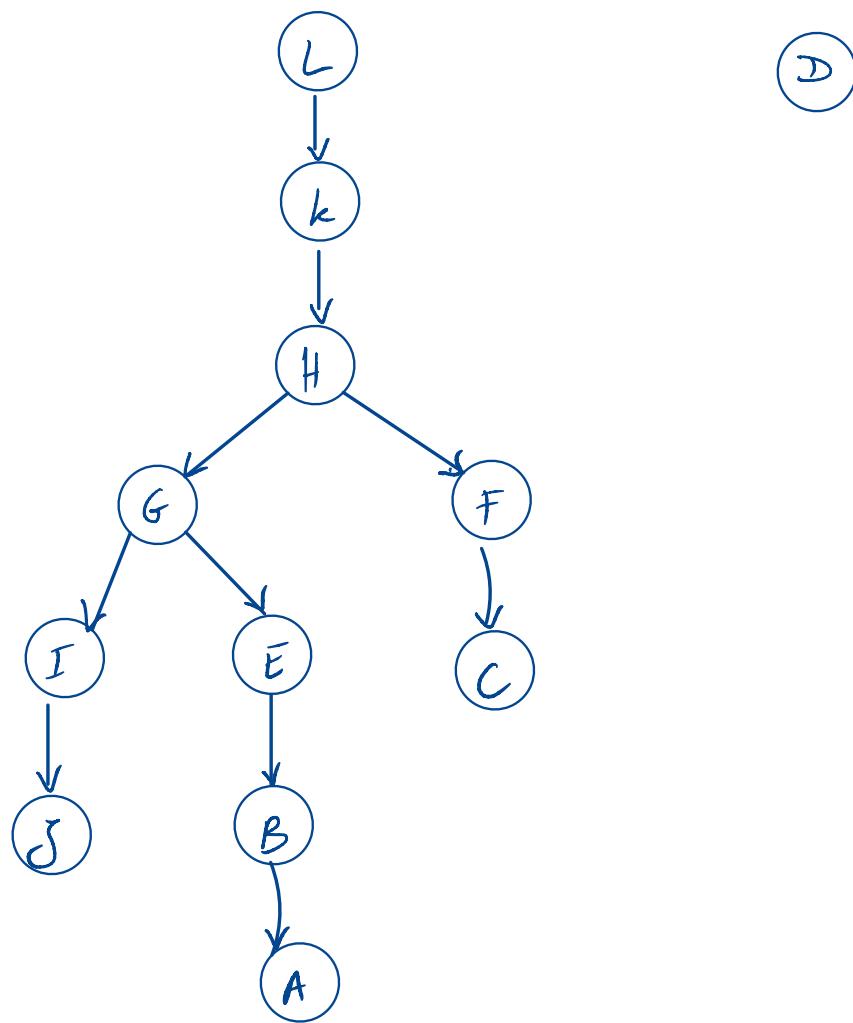
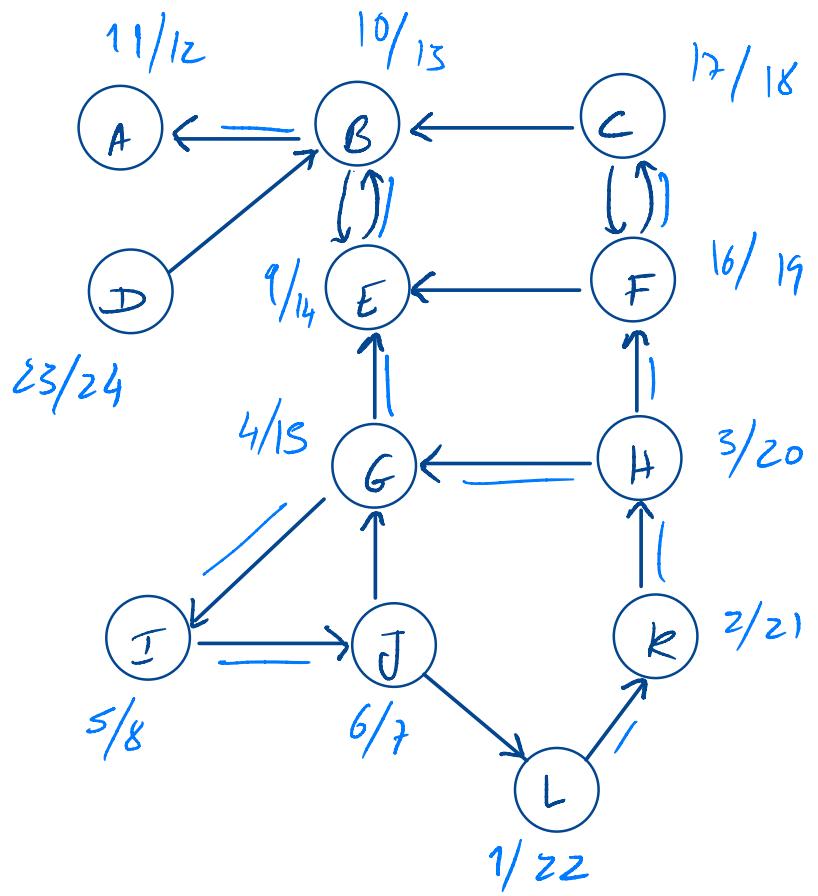
Aula 6

## Componentes Fortemente Ligados - Algoritmo de Kosaraju-Sharir

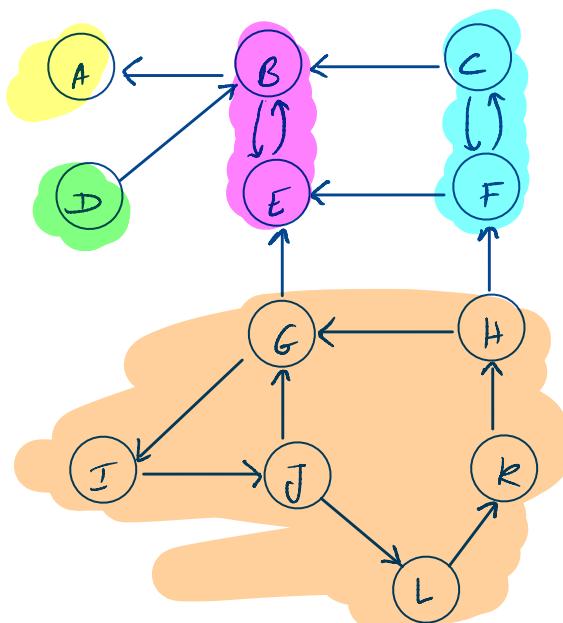
Floresta DFS



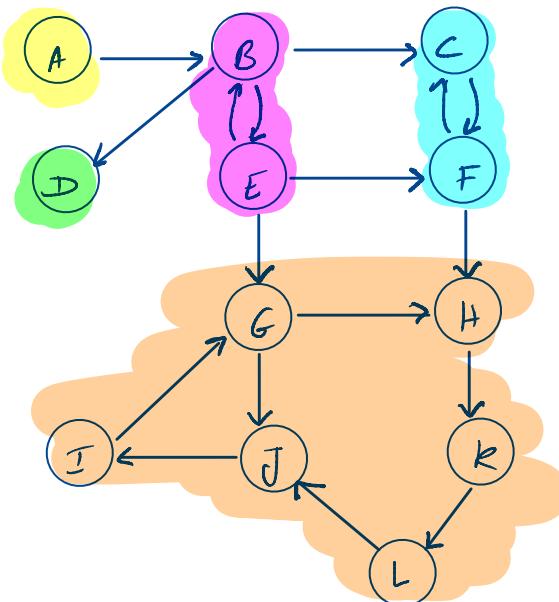
## Componentes Fortemente Ligados - Observações



## Componentes Fortemente Ligados - Grafos Transpostos



Grafo Original



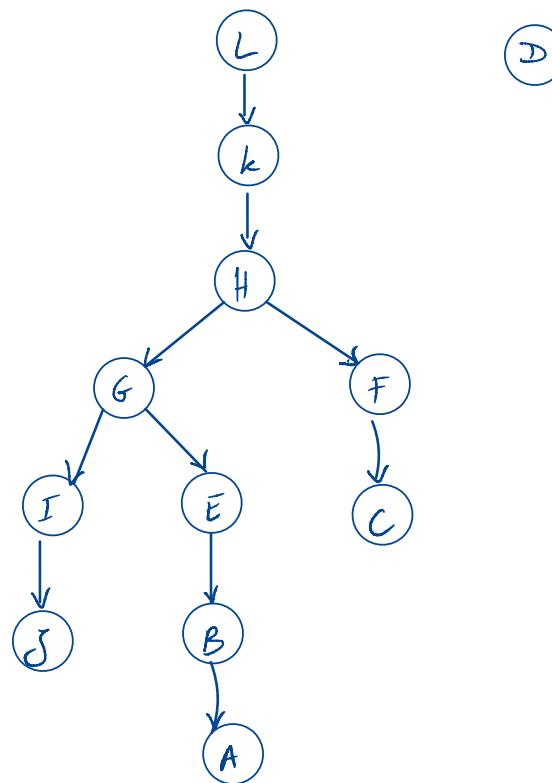
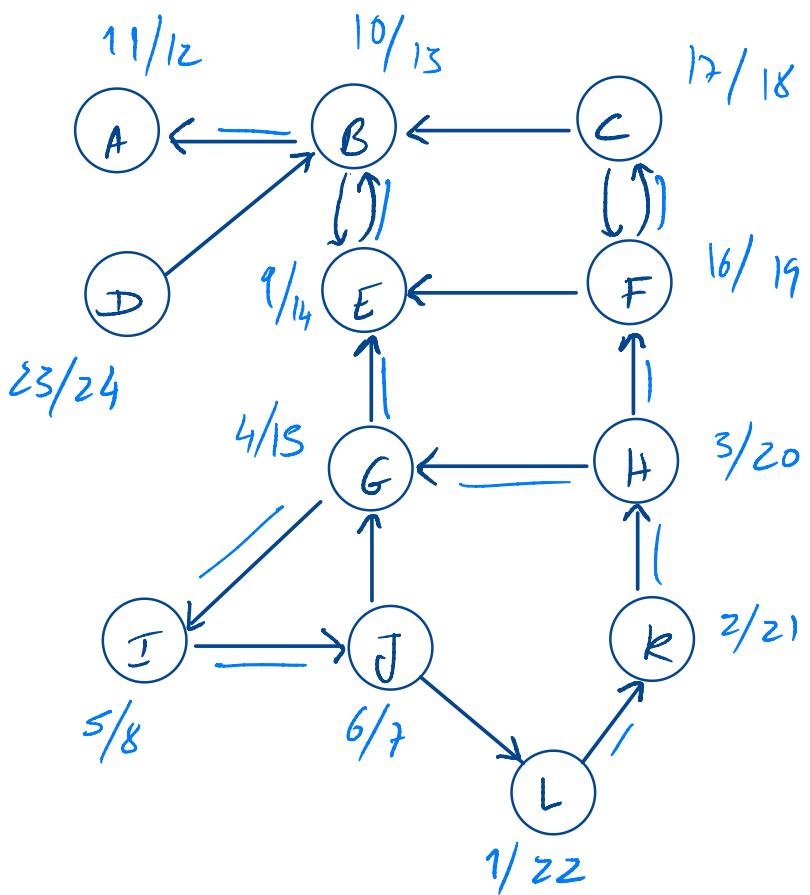
Grafo Transposto

Observação 1: Os SCCs do grafo original coincidem com os SCCs do grafo transposto

Observação 2:

- O componente com maior tempo de fim não tem arcos para trás no grafo transposto

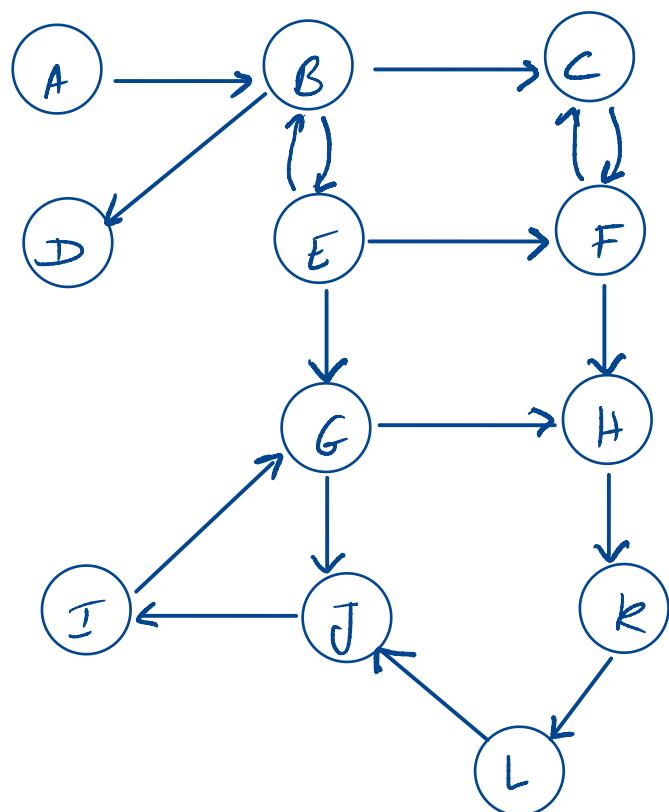
## Componentes Fortemente Ligados



Ordem Decrescente de tempo de fim

## Componentes Fortemente Ligados - Algoritmo de Kosaraju-Sharir

2º DFS:

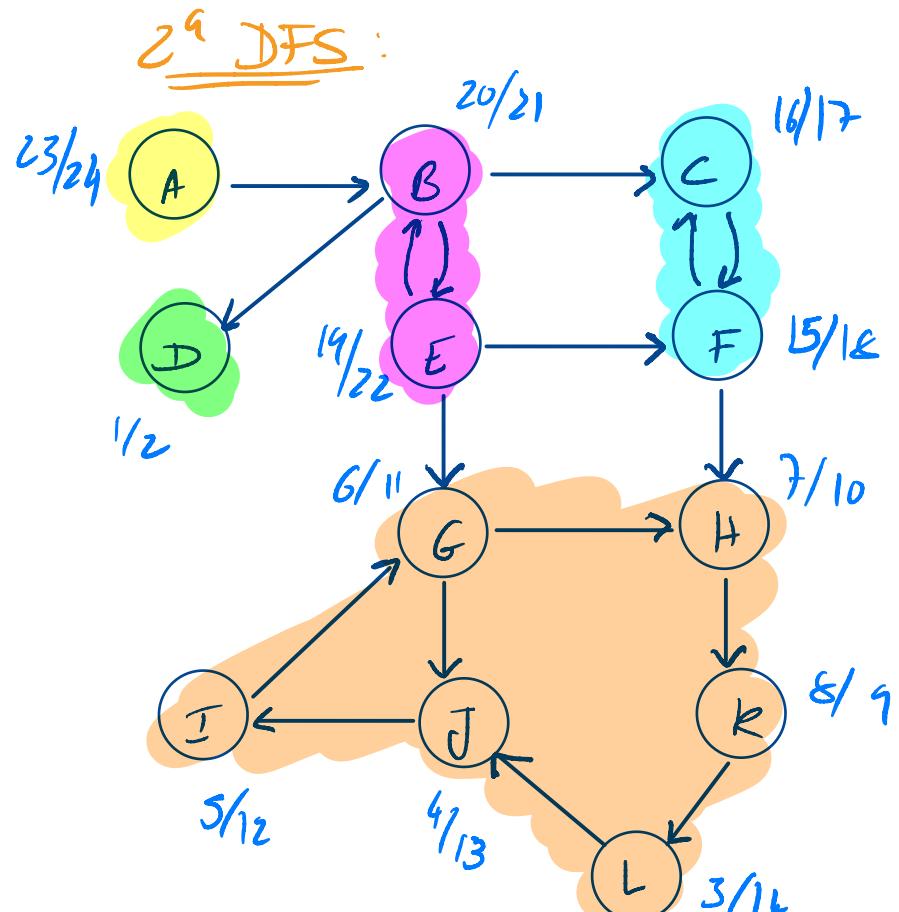


Ordem Decrescente de tempo de fim

D, L, K, H, F, C, G, E, B, A, I, J

Gráfico Tácos posto

## Componentes Fortemente Ligados - Algoritmo de Kosaraju-Sharir



Ordem Decrescente de tempo de fim

D, L, K, H, F, C, G, E, B, A, I, J	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
I	I	I	I	I	I	I	I	I	I	I	I

Grafo Tacos fasto

## Componentes Fortemente Ligados - Algoritmo de Kosaraju-Sharir

SCCs ( $G$ )

- Executar DFS ( $G$ ) para cálculo do tempo de fim
- Calcular  $G^T$
- Executar DFS ( $G^T$ ) escolhendo os nós por ordem inversa de tempo de fim na 1<sup>a</sup> DFS
  - O conjunto dos vértices de cada DFS corresponde a um SCC

Complexidade:  $\underline{\underline{O(V+E)}}$   $\hookrightarrow$  complexidade de DFS

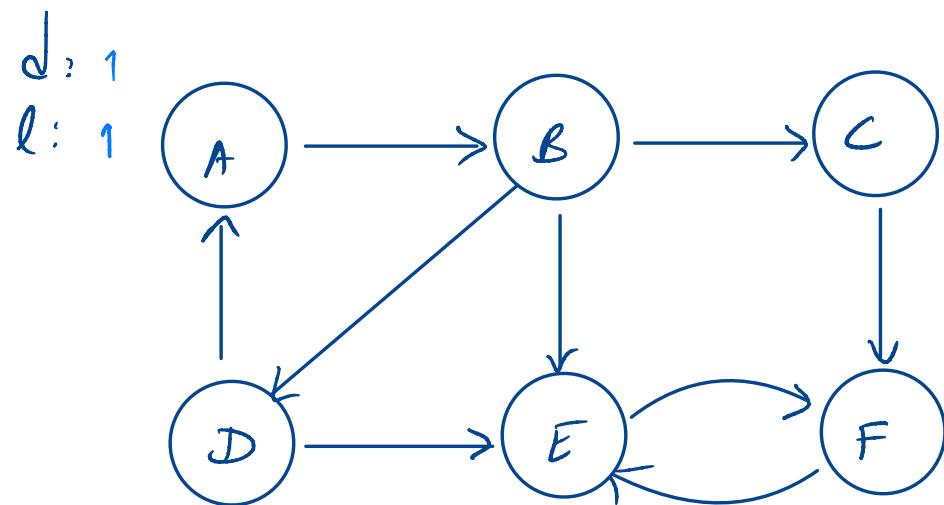
## Algoritmo de Tarjan

- Uma única travessia do grafo
- Traz passado da informação direto pelos arcos para trás e pelos arcos de cruzamento

$$\text{low}[u] = \min \left\{ d[v] \mid v \text{ é atingível a partir de } u \right\}$$

(primeira tentativa)

$\equiv$

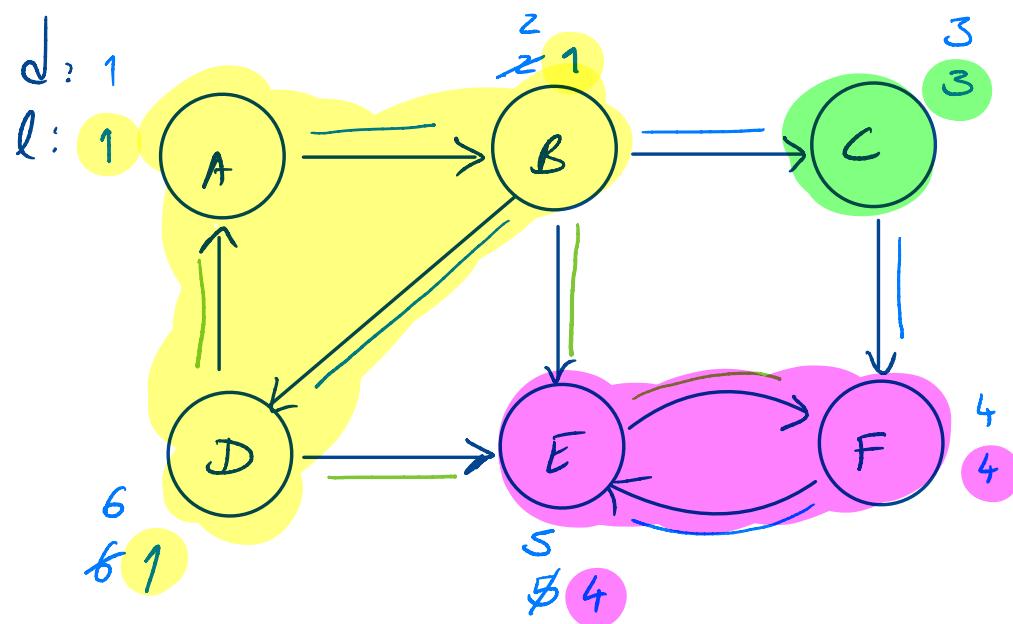


## Algoritmo de Tarjan

- Uma única travessia do grafo
- Tira o passado da informação através pelos arcos para trás e pelos arcos de cruzamento

$$\text{low}[u] = \min \left\{ d[v] \mid v \text{ é atingível a partir de } u \right\}$$

(primeira tentativa)  
=



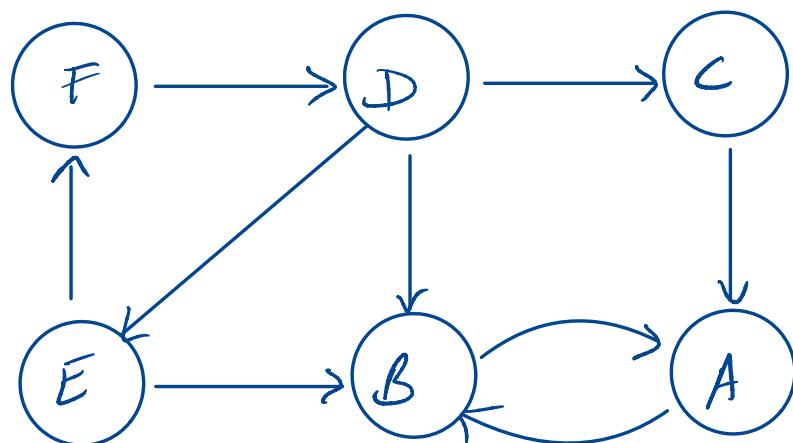
## Algoritmo de Tarjan

- Uma única travessia do grafo
- Tira o passado da informação através pelos arcos para trás e pelos arcos de cruzamento

$$\text{low}[u] = \min \left\{ d[v] \mid v \text{ é atingível a partir de } u \right\}$$

(primeira tentativa)

$\equiv$

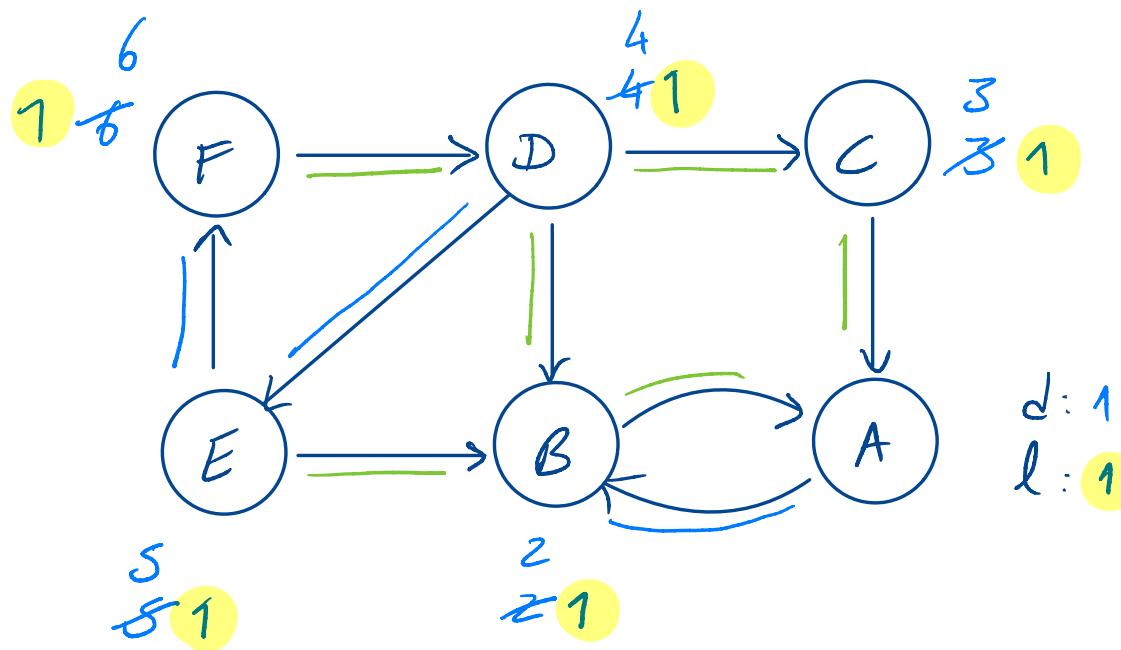


## Algoritmo de Tarjan

- Uma única travessia do grafo
- Trazem para trás da informação vindos pelos arcos para trás e pelos arcos de cruzamento

$$\text{low}[u] = \min \left\{ d[v] \mid v \text{ é atingível a partir de } u \right\}$$

(primeira tentativa)



\* Neste caso os tempos de low não nos ajudam.

↓  
Temos de mudar a definição

## Algoritmo de Tarjan

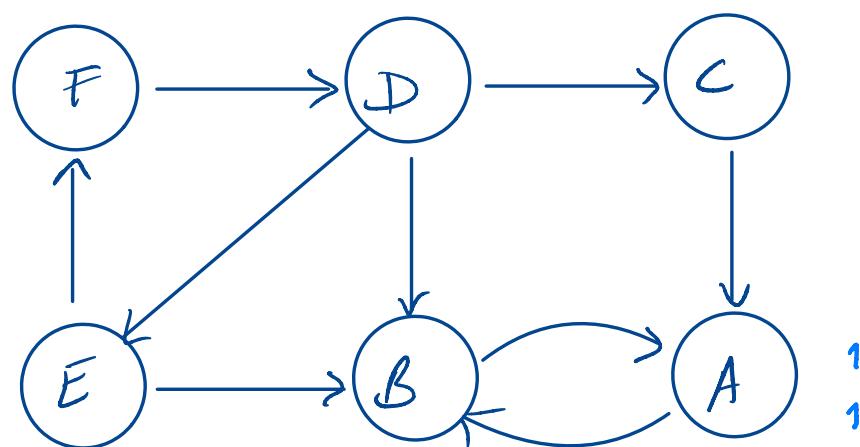
$low[u] = \min \{ d[v] \mid v \text{ é atingível a partir de } u$   
sem passar pelos SCCs já descobertos  
usando apenas um anel de pilhas ou de cavaamento }

- Como é que eu sei que já acabei de explorar um SCC?
  - Manter uma pilha com os nós que estamos a explorar
  - Quando fechamos um nó  $u$  com  $d[u] = low[u]$ , concluímos que encontrámos um SCC.
  - Os vértices do SCC são os nós em cima de  $u$  na pilha.

## Algoritmo de Tarjan

$low[u] = \min \{ d[v] \mid v \text{ é atingível a partir de } u$   
sem passar pelos SCCs já descobertos }

- Como é que eu sei que já acabei de explorar um SCC?



Pilha

1  
1

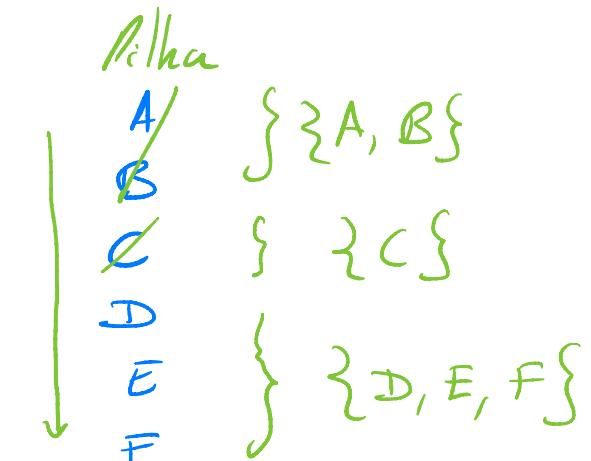
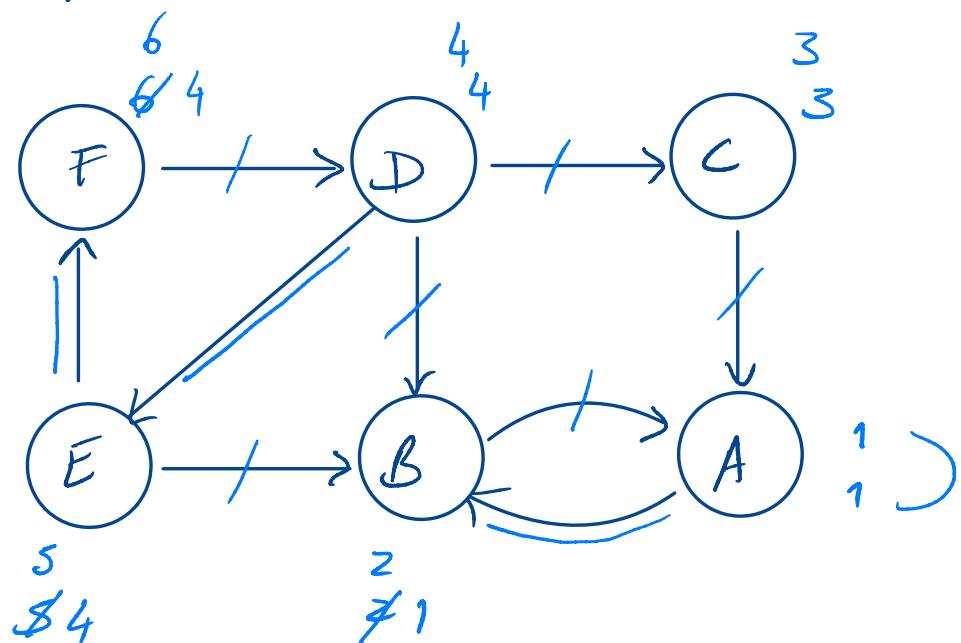
## Algoritmo de Tarjan

$low[u] = \min \{ d[v] \}$  é atingível a partir de u

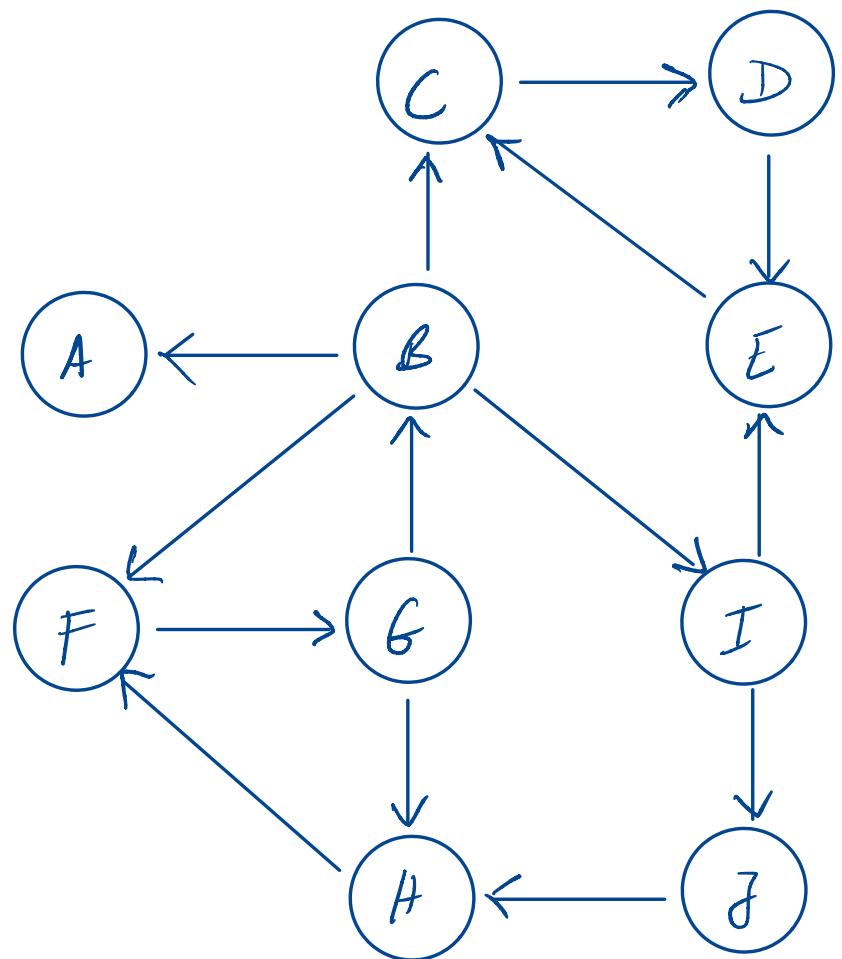
sem passar pelos SCCs já descobertos

e usando apenas um arco para trás ou de cruzamento

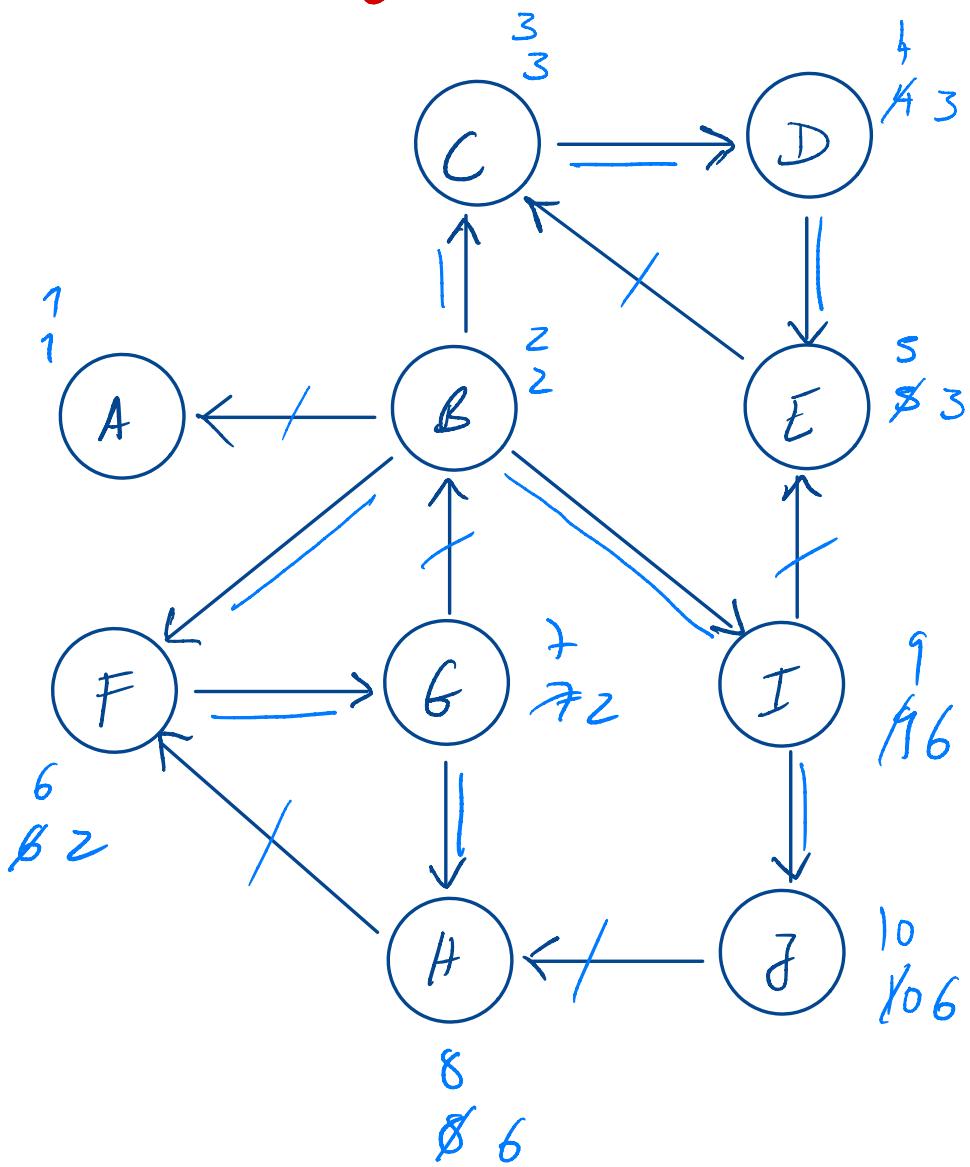
- Como é que eu sei que já acabei de explorar um SCC?



# Algoritmo de Tarjan



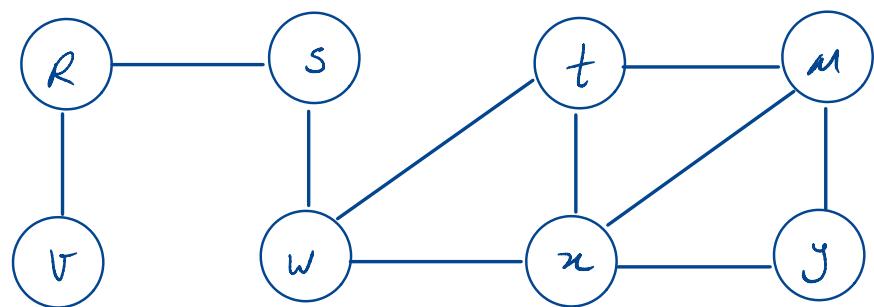
# Algoritmo de Tarjan



L:

{ {C, D, E} }  
{ {B, F, G, H, I, J} }

## Procura em Largura Primeiro (Breadth First Search - BFS)



Definição:

$\delta(s, u)$ : nº de aços do caminho  
mais curto entre  $s$  e  $u$

Propriedade [Desigualdade Triangular]

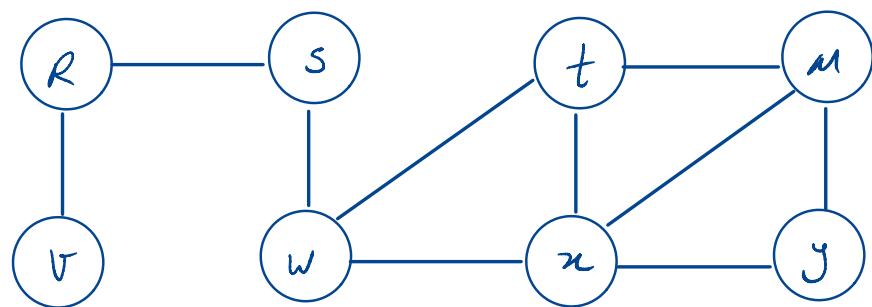
$$(u, v) \in E \Rightarrow \delta(s, r) \leq \delta(s, u) + 1$$

Exemplo:

$$\cdot \delta(s, y) ?$$

$$\cdot \delta(s, x) ?$$

## Procura em Largura Primeiro (Breadth First Search - BFS)



Objetivo:

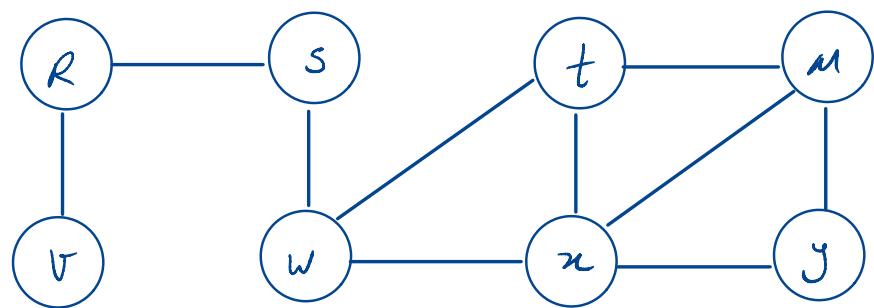
- Calcular para todos os vértices  $v \in V$ :  $\underline{d}(s, v) = \underline{S}(s, v)$

- Calculamos para cada vértice  $u \in V$ :

-  $d[u]$ : estimativa de  $S(s, u)$   $\rightarrow$  No fim do algoritmo  $d[v] = S(s, v)$

-  $\pi[u]$ : pai do vértice  $u$  na árvore BFS

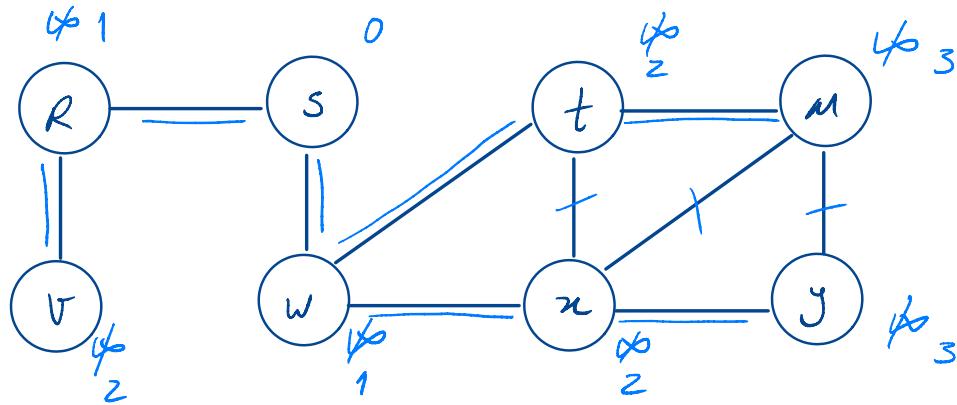
## Procura em Largura Primeiro (Breadth First Search - BFS)



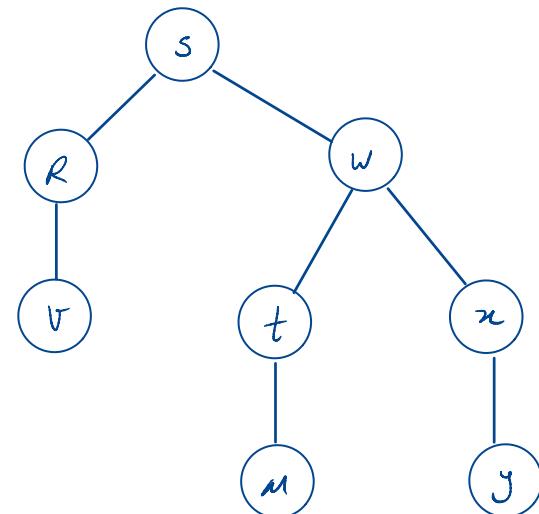
- Calculamos para cada vértice  $u \in V$ :
  - $d[u]$ : estimativa de  $\delta(s, v)$
  - $\pi[u]$ : pai do vértice  $u$  na árvore BFS

- $\text{color}[u]$ : estado de  $u$  na BFS
  - White: não visitado
  - Black: já visitado
  - Gray: marcado para visita

Procura em Largura Primeiro  
(Breadth First Search - BFS)



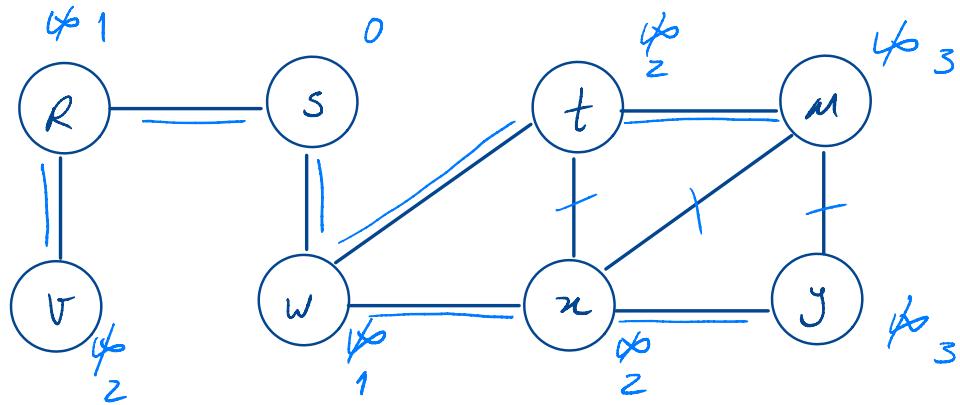
Árvore BFS



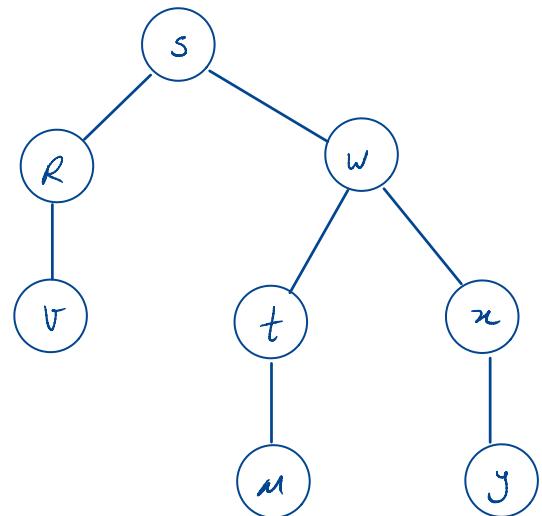
- Calculamos para cada vértice  $u \in V$ :
  - $d[u]$ : estimativa de  $s(s, u)$
  - $\pi[u]$ : pai do vértice  $u$  na árvore BFS

$$G_{\pi}^s = (V_{\pi}, E_{\pi})$$

Procura em Largura Primeiro  
(Breadth First Search - BFS)



Árvore BFS



- Calculamos para cada vértice  $u \in V$ :
  - $d[u]$ : estimativa de  $\delta(s, u)$
  - $\pi[u]$ : pai do vértice  $u$  na árvore BFS

$$G_\pi^s = (V_\pi, E_\pi)$$

$$V_\pi = \{r \mid \pi[r] \neq s\} \cup \{s\}$$

$$E_\pi = \{(\pi[v], v) \mid v \in V_\pi \setminus \{s\}\}$$

## Procura em Largura Primeiro (Breadth First Search - BFS)

BFS( $G, s$ )

for each  $v \in G.V \setminus \{s\}$

} set up dos vértices  
(d, π, color)

$Q := \text{NewQueue}(); Q.\text{enqueue}(s);$

} set up da fila que faz  
a gestão dos vértices

while ( $\neg Q.\text{empty}()$ ) {

} Loop principal

|      $v := Q.\text{dequeue}();$

}

## Program em Linguagem Primeiro (Breadth First Search - BFS)

BFS( $G, s$ )

for each  $v \in G.V \setminus \{s\}$

$v.\pi := \text{Nil}$ ;  $v.\text{color} := \text{White}$ ;  $v.d := +\infty$ ;

$s.\pi := \text{Nil}$ ;  $s.\text{color} := \text{Gray}$ ;  $s.d := 0$ ;

$Q := \text{NewQueue}(); Q.\text{enqueue}(s);$

while ( $\text{! } Q.\text{empty}()$ ) {

|  $v := Q.\text{dequeue}();$

| for each  $w \in G.\text{Adj}[v]$

| | if  $w.\text{color} == \text{White}$

| | |  $Q.\text{enqueue}(w)$

| | |  $w.\text{color} := \text{Gray}$ ;  $w.\pi := v$ ;  $w.d := v.d + 1$

| |  $v.\text{color} := \text{Black}$

} set up dos vértices  
( $d, \pi, \text{color}$ )

} set up da fila que faz  
a gestão dos vértices

} loop principal

## Procura em Largura Primeiro - Complexidade

BFS( $G, s$ )

for each  $v \in G.V \setminus \{s\}$

$v.\pi := \text{Nil}$ ;  $v.\text{color} := \text{White}$ ;  $v.d := +\infty$ ;

$s.\pi := \text{Nil}$ ;  $s.\text{color} := \text{Gray}$ ;  $s.d := 0$ ;

$Q := \text{NewQueue}(); Q.\text{enqueue}(s);$

while ( $\neg Q.\text{empty}()$ ) {

|  $v := Q.\text{dequeue}();$

| for each  $w \in G.\text{Adj}[v]$

| | if  $w.\text{color} == \text{White}$

| | |  $Q.\text{enqueue}(w)$

| | |  $w.\text{color} := \text{Gray}$ ;  $w.\pi := v$ ;  $w.d := v.d + 1$

| |  $v.\text{color} := \text{Black}$

| } ↳ Loop 2

↳ Loop 1

Análise de complexidade:

- Cada vértice é inserido no máximo

- O loop 1 é executado no máximo

- O loop 2 é executado no máximo

- Complexidade:

## Procura em Largura Primeiro - Complexidade

BFS( $G, s$ )

for each  $v \in G.V \setminus \{s\}$

$v.\pi := \text{Nil}$ ;  $v.\text{color} := \text{White}$ ;  $v.d := +\infty$ ;

$s.\pi := \text{Nil}$ ;  $s.\text{color} := \text{Gray}$ ;  $s.d := 0$ ;

$Q := \text{NewQueue}(); Q.\text{enqueue}(s);$

while ( $\neg Q.\text{empty}()$ ) {

|  $v := Q.\text{dequeue}();$

| for each  $w \in G.\text{Adj}[v]$

| | if  $w.\text{color} == \text{White}$

| | |  $Q.\text{enqueue}(w)$

| | |  $w.\text{color} := \text{Gray}$ ;  $w.\pi := v$ ;  $w.d := v.d + 1$

| |  $v.\text{color} := \text{Black}$

| } ↳ Loop 2

↳ Loop 1

→ depois de passar  
a cinzento um  
vértice nunca mais  
se torna branco

Análise de complexidade:

- Cada vértice é inserido no máximo uma vez na fila de prioridade
- O loop 2 é executado no máximo uma vez por cada arco do grafo
- Complexidade:  $O(V+E)$

## Procura em Largura Primeiro - Invariante de Distâncias

BFS( $G, s$ )

for each  $v \in G.V \setminus \{s\}$

$v.\pi := \text{Nil}$ ;  $v.\text{color} := \text{White}$ ;  $v.d := +\infty$ ;

$s.\pi := \text{Nil}$ ;  $s.\text{color} := \text{Gray}$ ;  $s.d := 0$ ;

$Q := \text{NewQueue}(); Q.\text{enqueue}(s);$

while ( $\neg Q.\text{empty}()$ ) {

|  $v := Q.\text{dequeue}();$

| for each  $w \in G.\text{Adj}[v]$

| | if  $w.\text{color} == \text{White}$

| | |  $Q.\text{enqueue}(w)$

| | |  $w.\text{color} := \text{Gray}$ ;  $w.\pi := v$ ;  $w.d := v.d + 1$

| |  $v.\text{color} := \text{Black}$

| }  $\hookrightarrow \text{Loop 2}$

$\hookrightarrow \text{Loop 1}$

I  $\forall v. d[v] \leq s(s, v)$

II  $Q = \langle v_1, \dots, v_n \rangle$

# Procurar em Largura Primeiro - Invariante de Distâncias

BFS( $G, s$ )

for each  $v \in G.V \setminus \{s\}$

$v.\pi := \text{Nil}$ ;  $v.\text{color} := \text{White}$ ;  $v.d := +\infty$ ;

$s.\pi := \text{Nil}$ ;  $s.\text{color} := \text{Gray}$ ;  $s.d := 0$ ;

$Q := \text{NewQueue}(); Q.\text{enqueue}(s);$

while ( $\neg Q.\text{empty}()$ ) {

$v := Q.\text{dequeue}();$

    for each  $w \in G.\text{Adj}[v]$

        if  $w.\text{color} == \text{White}$

$Q.\text{enqueue}(w)$

$w.\text{color} := \text{Gray}$ ;  $w.\pi := v$ ;  $w.d := v.d + 1$

$v.\text{color} := \text{Black}$

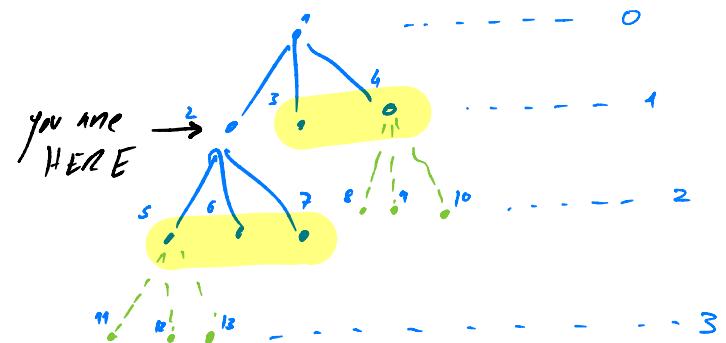
} Loop 2

Loop 1

①  $\forall v. d[v] \geq s(s, v)$

②  $Q = \langle v_1, \dots, v_n \rangle$

$d[v_1] \leq \dots \leq d[v_n] \leq d[v_1] + 1$



Provas para estudo individual

### Teorema 6.1

O algoritmo de Kosaraju-Sharir é correcto.

#### Prova:

- Provar a correctez do algoritmo de Kosaraju-Sharir implica mostrar que as árvores da floresta gerada pelo 2º DFS correspondem aos SCCs do grafo original.
- A prova faz-se por indução no nº n de árvores encontradas.  
Luso base n=0  $\Rightarrow$  Não há nenhuma prova.
- Luso Indutivo: Suponhamos que o algoritmo encontra (nH) árvores
  - Pela hipótese de indução concluímos que as n primeiras árvores são SCCs do grafo G. Temos de mostrar que a (n+1)-ésima primeira árvore também o é. Formalmente, seja  $T_{n+1}$  o conjunto dos vértices contidos na (n+1)-ésima árvore. Temos de mostrar que:  $T_{n+1}$  é um SCC.
  - Saja  $x$  o 1º vértice encontrado em  $T_{n+1}$ . Há que mostrar que:  $T_{n+1} = C_x \rightarrow$  componente que contém  $x$

• Mostreemos sofisticamente que:

$$T_{n+1} \subseteq C_x$$

$$C_x \subseteq T_{n+1}$$

(I)  $C_x \subseteq T_{n+1}$

Qd  $x$  é encontrado existe um caminho branco a ligar  $x$  a todos os vértices de  $C_x$ .

Logo, todos os vértices em  $C_x$  são descendentes de  $x$  na floresta DFS (Teorema do Círculo Branco).

(II)  $T_{n+1} \subseteq C_x$

Suponhamos por contradição que  $T_{n+1} \not\subseteq C_x$ .

Segue q existe um vértice  $y$  tal que  $y \in T_{n+1}$  e  $y \notin C_x$ .  
(Observamos q  $y$  tb não pertence aos componentes até aqui identificados).

Segue q:



em  $G'$



em  $G$

↓ Isto implica que  $f(y) > f(x)$

→ Contradiz a hipótese de que visitamos os vértices por ordem decrescente de tempo de fim.

### Teorema 6.2

O Invariante 2 da BFS é mantido pelo algoritmo.

### Prova

$$Q = \langle v_1, \dots, v_n \rangle \Rightarrow v_1.d \leq \dots \leq v_n.d \leq v_1.d + 1$$

- Queremos mostrar que o invariante se mantém quando retiramos o vértice  $v_i$  do início da fila e colocamos no fim da fila os seus sucessores brancos.
$$\text{I} \quad v_n.d \leq v_i.d + 1 = n.d \quad \checkmark$$
- Primeiro sucessor branco:  $m$ 
$$m.d = v_i.d + 1$$
$$Q' = \langle v_2, \dots, v_n, m \rangle$$
$$\text{II} \quad n.d \leq v_2.d + 1$$
$$v_1.d + 1 \leq v_2.d + 1$$
$$v_1.d \leq v_2.d \quad \checkmark$$

Há que mostrar que:

  - (I)  $n.d \geq v_n.d$
  - (II)  $n.d \leq v_2.d + 1$

### Teorema 6.3

O Invariante 1 da BFS é mantido pelo algoritmo.

### Prova

Início     $\forall v \neq s. \quad v.d = \infty \geq \delta(s, v)$   
 $s.d = 0 = \delta(s, s)$

### Passo

- O passo acontece quando ao visitar  $u$  actualizamos as distâncias dos seus vizinhos brancos.

Seja  $v$  o vizinho branco de  $u$ :

$$\begin{aligned} v.d &= u.d + 1 && \xrightarrow{\text{Invariante}} \\ &\geq \delta(s, u) + 1 && \xrightarrow{\text{Desigualdade}} \\ &\geq \delta(s, v) && \xleftarrow{\text{Triangular}} \end{aligned}$$

## Teorema 6.4 [Conexão BFS]

O algoritmo BFS é conexo.

Prova:

Quando o algoritmo  $BFS(G, s)$  termina temos  $\bar{g}$ :

$$\forall v \in V. \quad v.d = \delta(s, v)$$

- Provamos a conexão do algoritmo por contradição. Suponhamos  $\exists$  existe um nó  $m \in V$  que depois da execução de  $BFS(G, s)$ ,  $m.d \neq \delta(s, m)$ .
- Assumimos, sem perda de generalidade, que  $m$  é o primeiro vértice no caminho mais curto  $\bar{g}$  o liga a  $s$   $|k|$  cuja distância calculada não coincide c/ a distância mínima dada por  $\bar{s}$ .
- Seja  $w$  o predecessor de  $m$  no caminho mais curto que o liga a  $s$ , temos  $\bar{g} \quad w.d = \delta(s, w)$ .
- Quando  $w$  é explorado pela DFS, o vértice  $m$  pode ser: branco, cinzento, ou preto.  
Analizamos cada um das 3 casas separadamente.

[ $m$  é branco]  $m.d = w.d + 1 = \delta(s, w) + 1 = \delta(s, m)$

contradição

[ $m$  é perto] O vértice  $w$  foi retirado da fila, de onde segue pelo Invariante 2 que:

$$m \cdot d \leq w \cdot d = \delta(s, w) < \delta(s, m)$$

Combinando com o Invariante 1:

$$\delta(s, m) \leq m \cdot d < \delta(s, m) \quad \text{contradição}$$

[ $m$  é vizinho]  $w$  já se encontra na fila  $Q$  para ser explorado.

$$m \cdot d \leq w \cdot d + 1 = \delta(s, w) + 1 = \delta(s, m) \quad (\text{Invariante 2})$$

Se lemos pelo Invariante 1 que:  $m \cdot d \geq \delta(s, m)$ .

Assim concluímos que:

$$\delta(s, m) \leq m \cdot d \leq \delta(s, m)$$

De onde segue que  $\delta(s, m) = m \cdot d$ . contradição.