Aula 9

- Componentes Fortemente Ligados
- BFS

Componentes Fortemente Ligados

Definição [Componente Fortemente Ligado (Strongly Connected Component - SCC)]

Dado um grafo $G = (V, E)$, um componente fortemente ligado de G é um conjunto de vértices $C \subseteq V$ tal que:

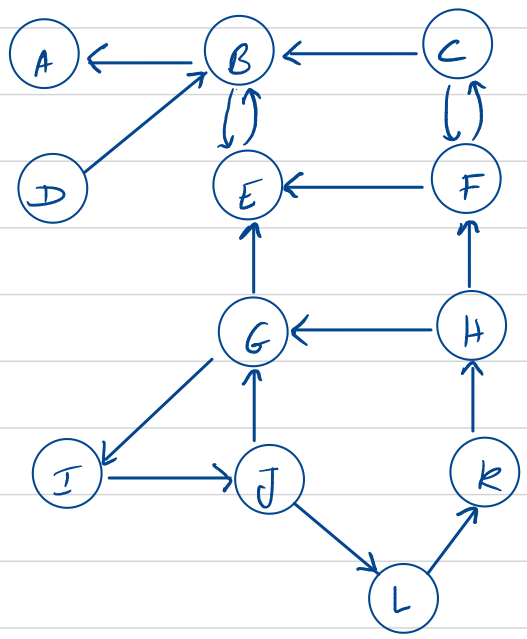
(I) $\forall u, v \in C.\ u \rightsquigarrow v \wedge v \rightsquigarrow u$

↳ u é atingível a partir de v

(II) C é maximal

(Não existe C' tal que $C \subset C'$ e C' satisfaz (I))

Exemplo:



Componentes Fortemente Ligados

Definição [Componente Fortemente Ligado (Strongly Connected Component - SCC)]

Dado um grafo $G=(V,E)$, um componente fortemente ligado de G é um conjunto de vértices $C \subseteq V$ tal que:

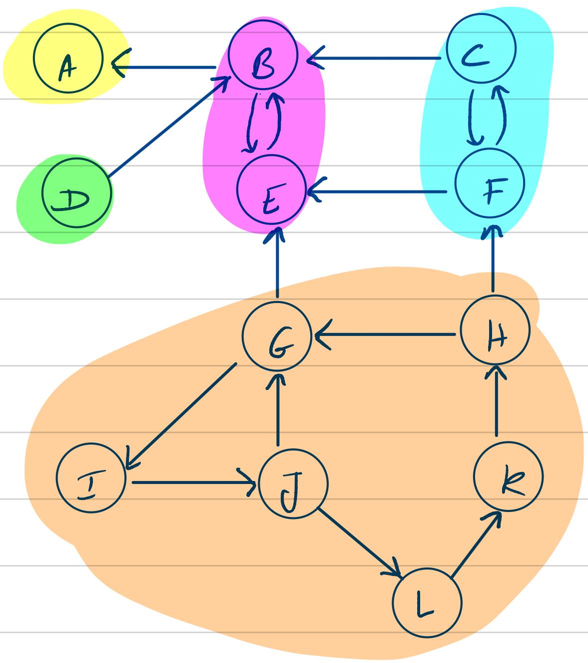
(I) $\forall u, v \in C.\ u \rightsquigarrow v \wedge v \rightsquigarrow u$

↳ u é atingível a partir de v

(II) C é maximal

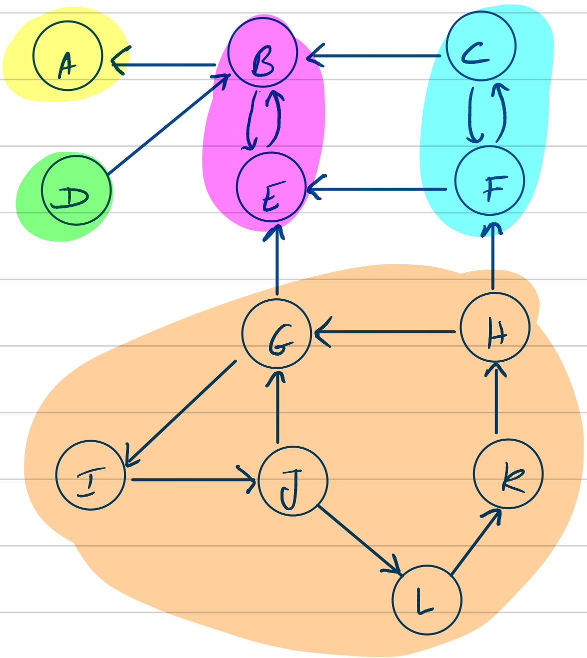
(Não existe C' tal que $C \subset C'$ e C' satisfaz (I))

Exemplo:



Componentes Fortemente Ligados

Exemplo:



Grafos dos SCCs:

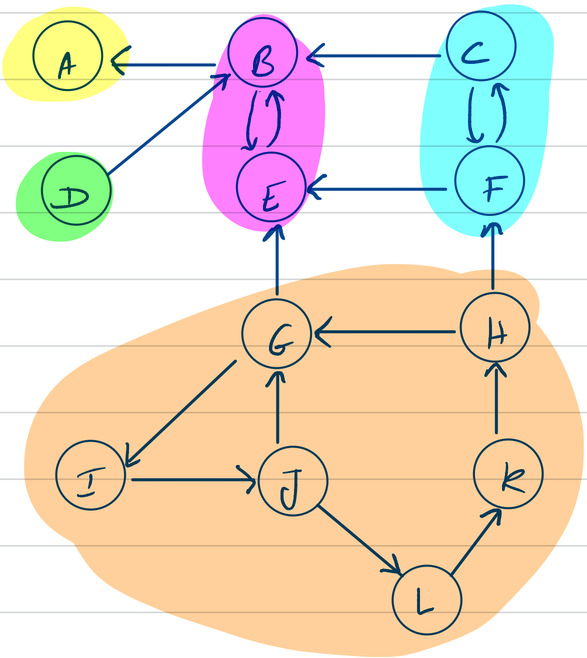
$$G_{SCC} = (V_{SCC}, E_{SCC})$$

$V_{SCC} =$

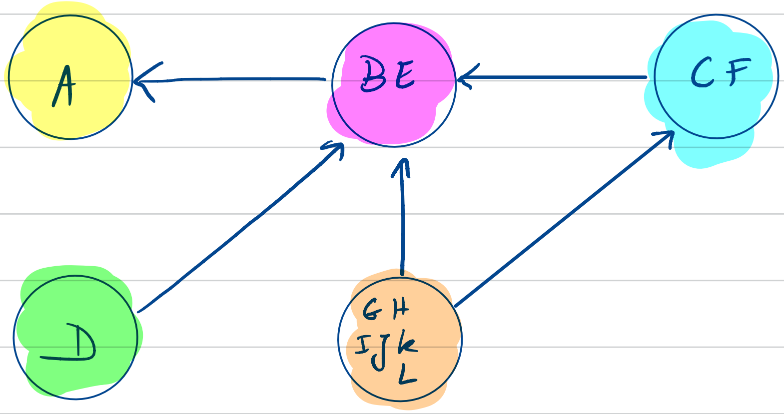
E_{SCC}:

Componentes Fortemente Ligados

Exemplo:



Grafos dos SCCs:



$$G_{SCC} = (V_{SCC}, E_{SCC})$$

$$V_{SCC} = \{ C \mid C \text{ é um SCC de } G \}$$

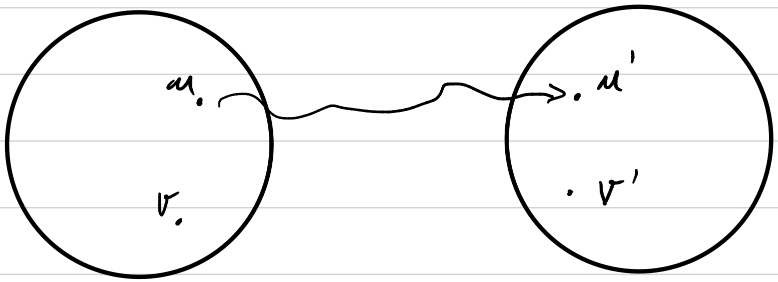
E_{SCC}: $(C_1, C_2) \in E_{SCC}$

sse

$\exists u, v.\ u \in C_1 \wedge v \in C_2 \wedge (u, v) \in E$

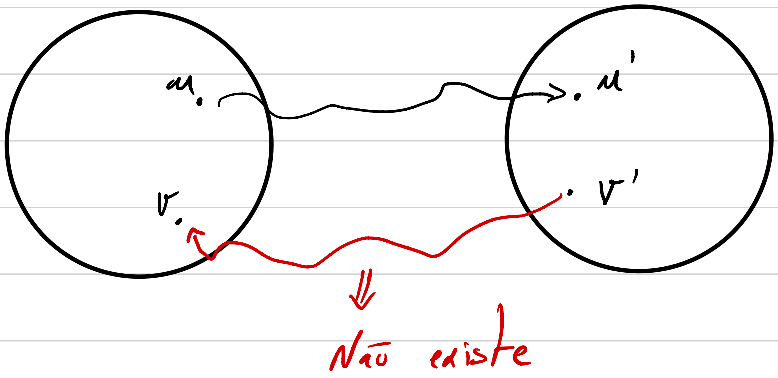
Componentes Fortemente Ligados - Propriedades

Propriedade 1: Sejam C e C' dois SCCs de um grafo $G = (V, E)$.
Dados dois vértices $u, v \in C$ e dois vértices $u', v' \in C'$, se $u \rightsquigarrow u'$
então $v' \not\rightsquigarrow v$.



Componentes Fortemente Ligados - Propriedades

Propriedade 1: Sejam C e C' dois SCCs de um grafo $G = (V, E)$.
Dados dois vértices $u, v \in C$ e dois vértices $u', v' \in C'$, se $u \rightsquigarrow u'$
então $v' \not\rightsquigarrow v$.



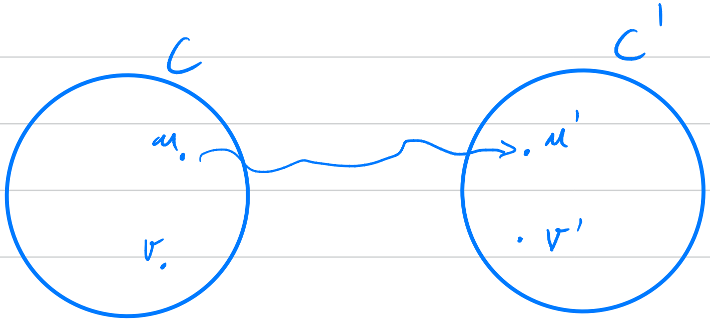
Componentes Fortemente Ligados - Propriedades

Propriedade 2: Sejam C e C' dois SCCs num grafo dirigido $G=(V,E)$; se existir um arco (u,u') de C para C', então:

$$f(C) > f(C')$$

- $d(C) = \min\{d(u) \mid u \in C\}$
- $f(C) = \max\{f(u) \mid u \in C\}$

Prova:



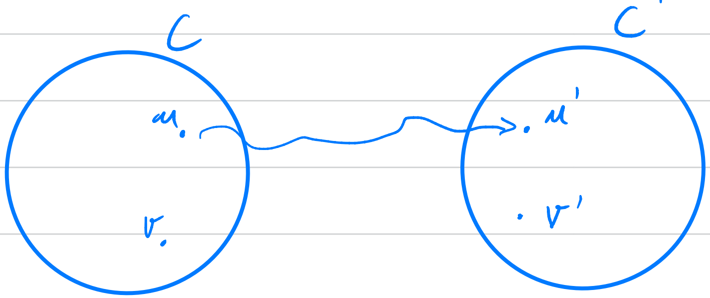
Componentes Fortemente Ligados - Propriedades

Propriedade 2: Sejam C e C' dois SCCs num grafo dirigido $G=(V,E)$; se existir um arco (u,u') de C para C', então:

$$f(C) > f(C')$$

- $d(C) = \min\{d(u) \mid u \in C\}$
- $f(C) = \max\{f(u) \mid u \in C\}$

Prova:



(I) $d_C < d_{C'}$

- Há um caminho branco q̄ liga u a todos os vértices de C'. Todos os vértices de C' são descendentes de u.

- $f(C') < f(C)$

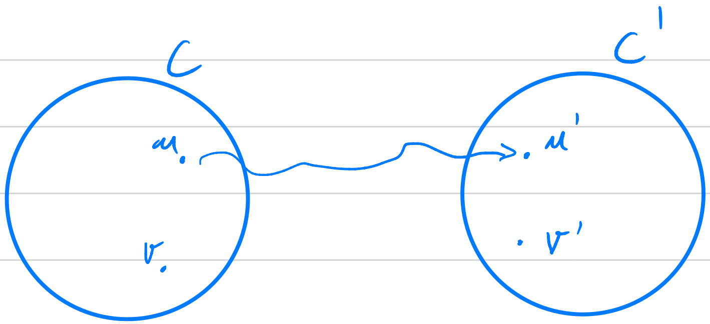
Componentes Fortemente Ligados - Propriedades

Propriedade 2: Sejam C e C' dois SCCs num grafo dirigido $G=(V,E)$; se existir um arco (u,u') de C para C', então:

$$f(C) > f(C')$$

- $d(C) = \min\{d(u) \mid u \in C\}$
- $f(C) = \max\{f(u) \mid u \in C\}$

Prova:



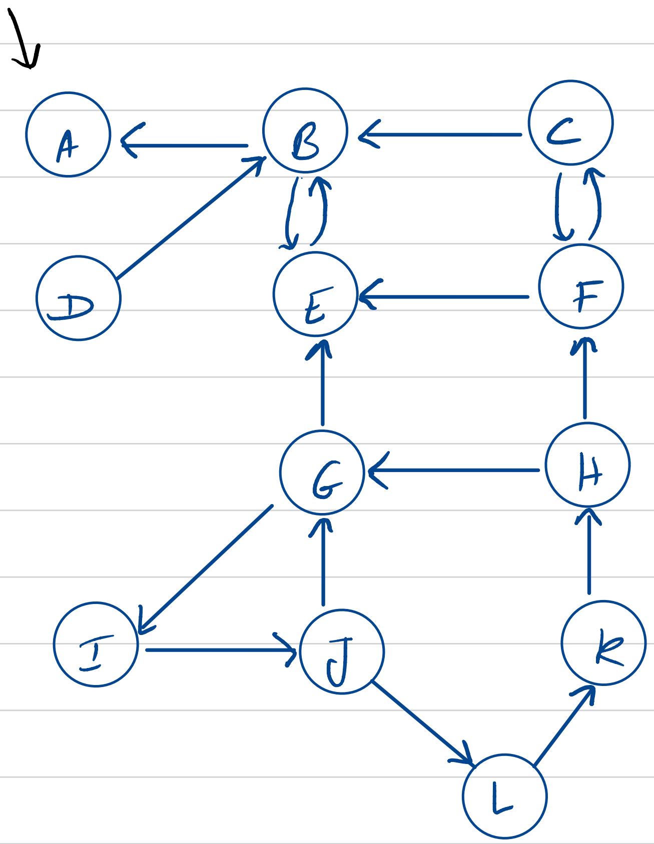
(I) $d(C') < d(C)$

- Não existe nenhum arco de C' p/ C
- Não existe um caminho a ligar os vértices de C' aos vértices de C.

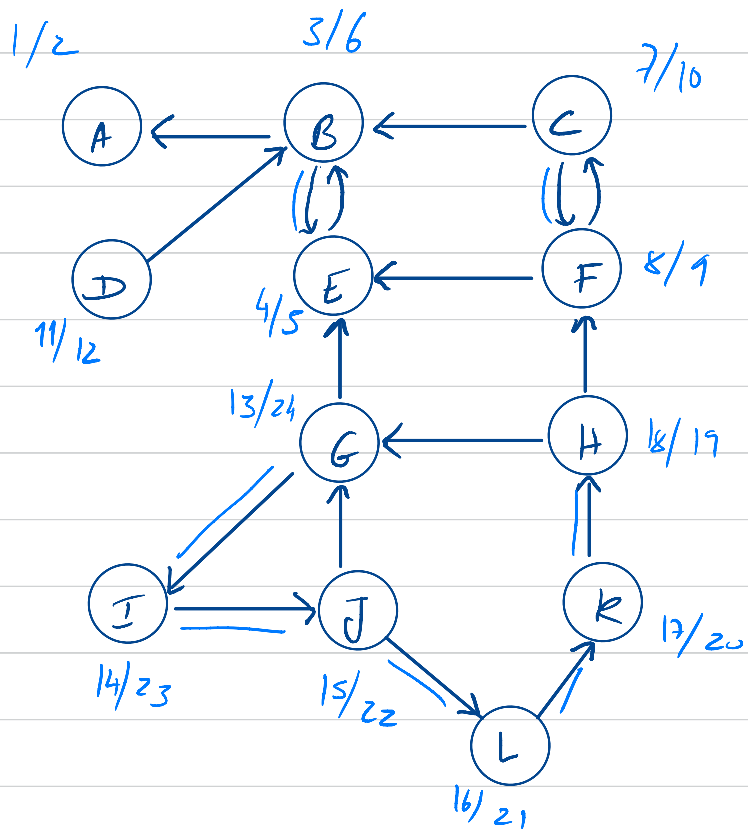
- Todos os vértices de C' são fechados antes de os vértices de C começarem a ser explorados.

$$f(C') < d(C) < f(C)$$

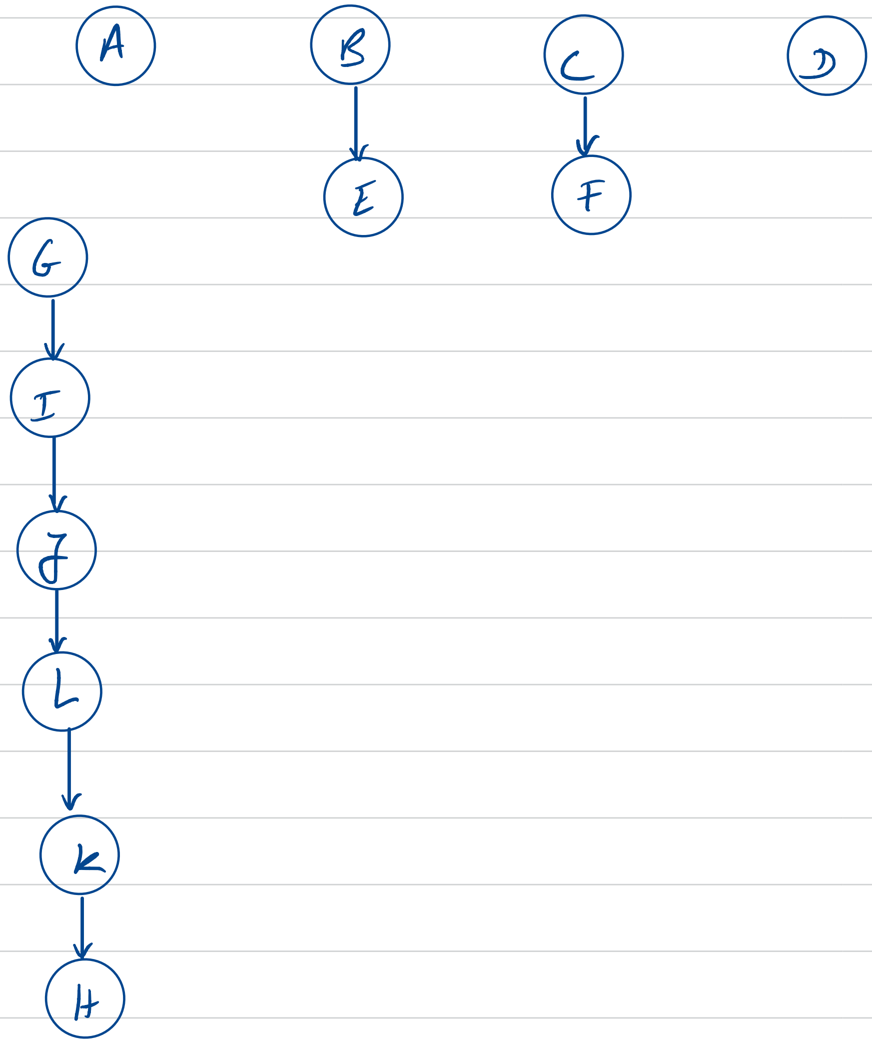
Componentes Fortemente Ligados - Algoritmo de Kosaraju-Sharir



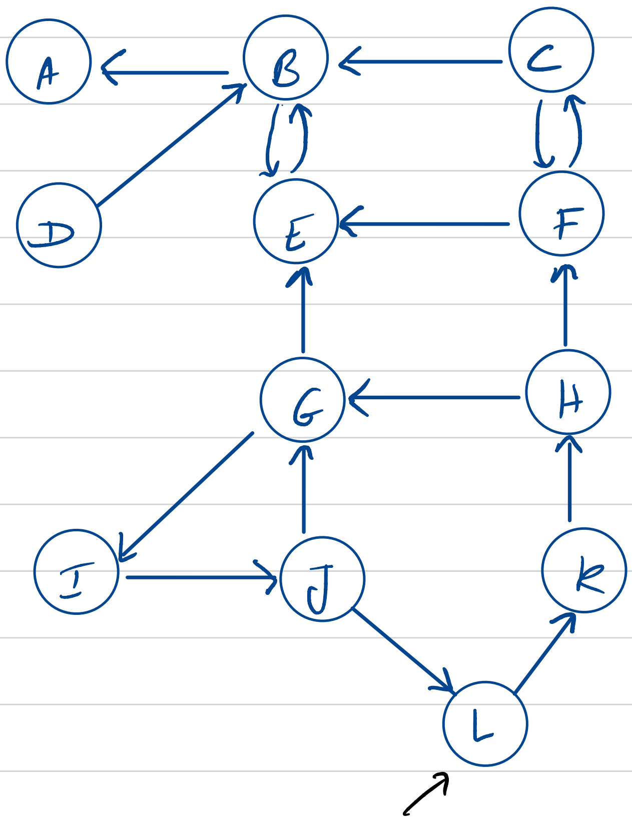
Componentes Fortemente Ligados - Algoritmo de Kosaraju - Sharir



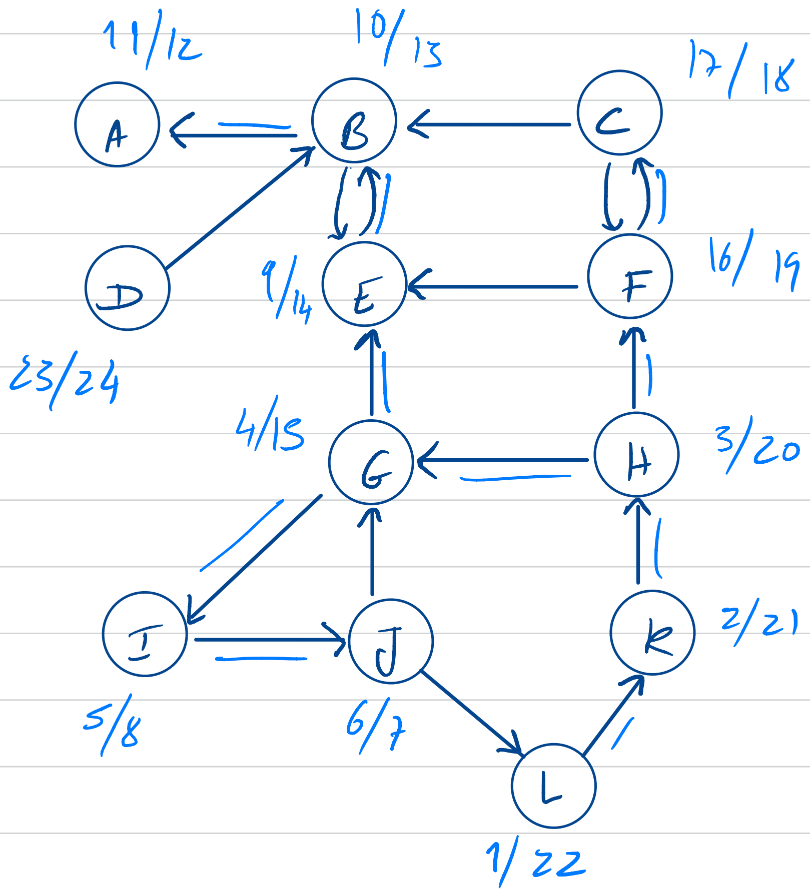
Floresta DFS

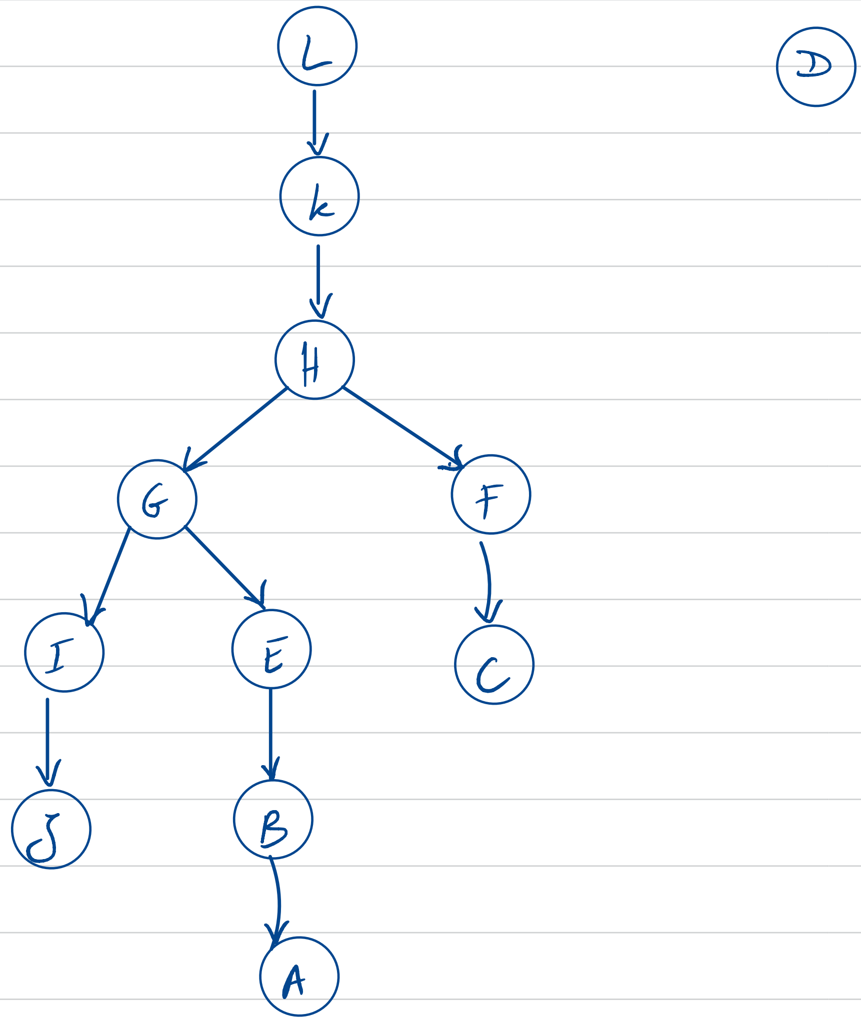


Componentes Fortemente Ligados - Algoritmo de Kosaraju-Sharir

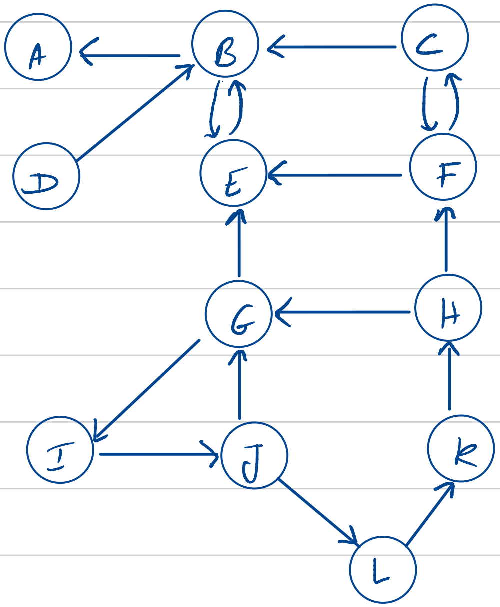


Componentes Fortemente Ligados - Observações

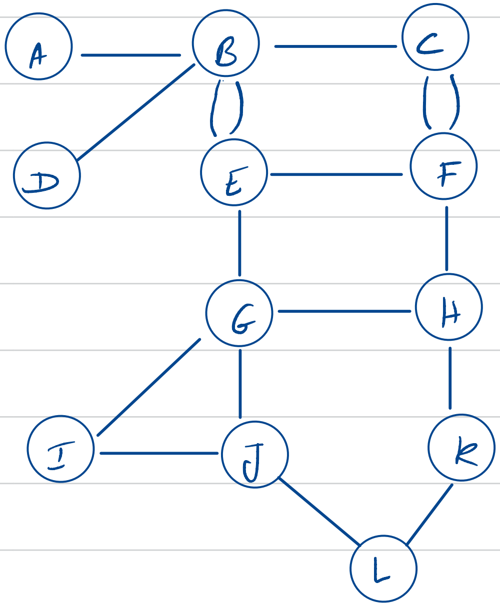




Componentes Fortemente Ligados - Grafos Transpostos



Grafo Original



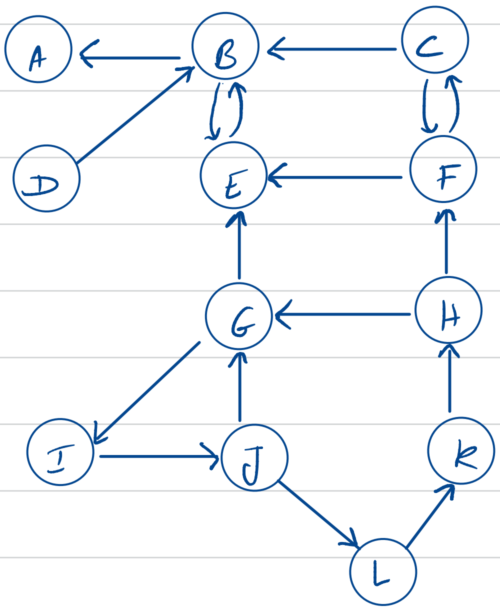
Grafo Transposto

Grafo Transposto

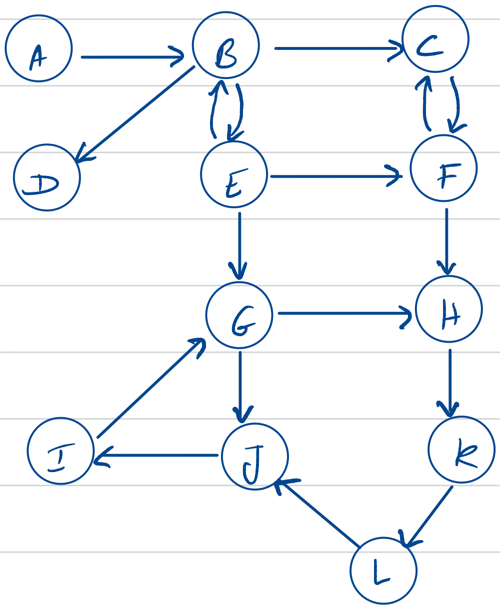
$G^T = (V, E^T)$

$E^T = \{ (u, v) \mid (v, u) \in E \}$

Componentes Fortemente Ligados - Grafos Transpostos

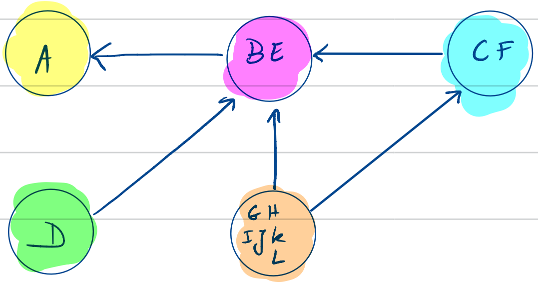


Grafo Original



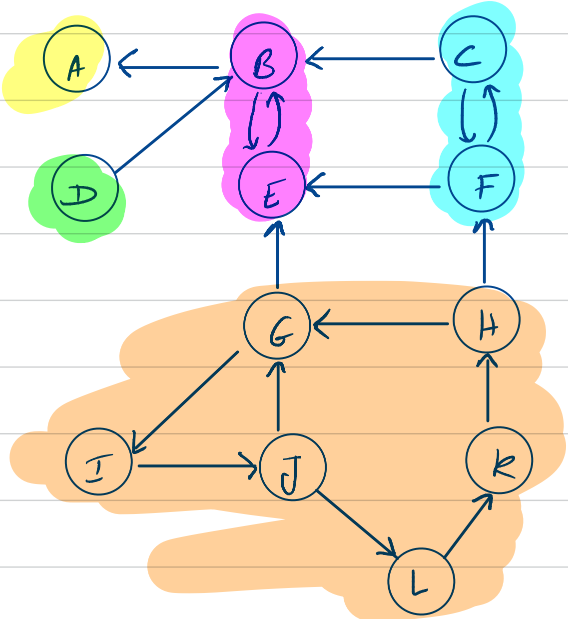
Grafo Transposto

Grafo Original

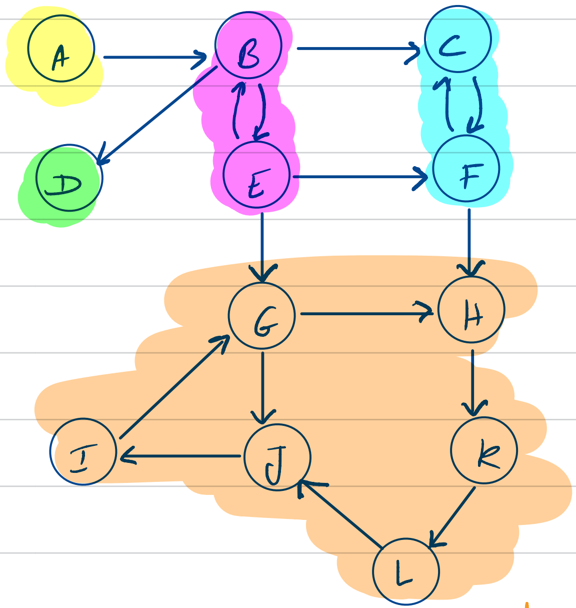


Grafo Transposto

Componentes Fortemente Ligados - Grafos Transpostos

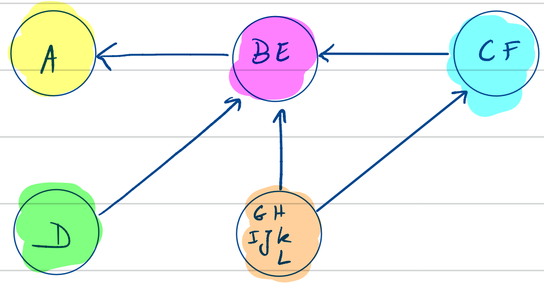


Grafo Original

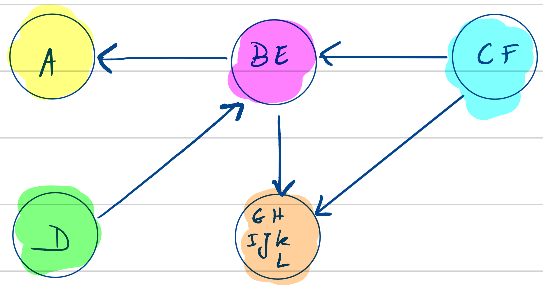


Grafo Transposto

Grafo Original



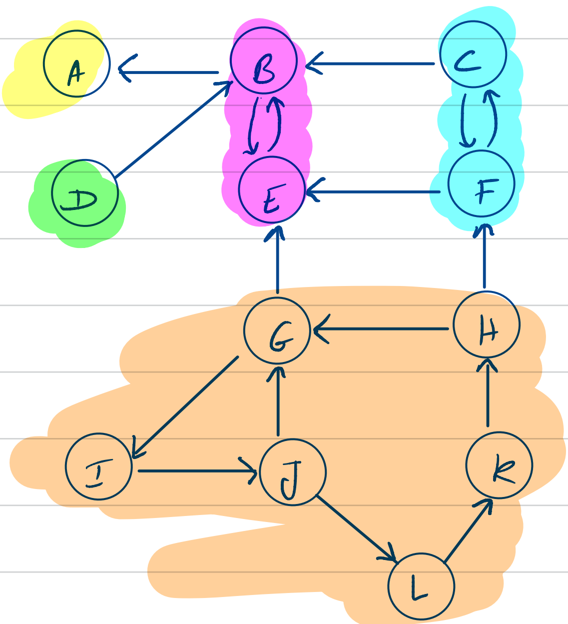
Grafo Transposto



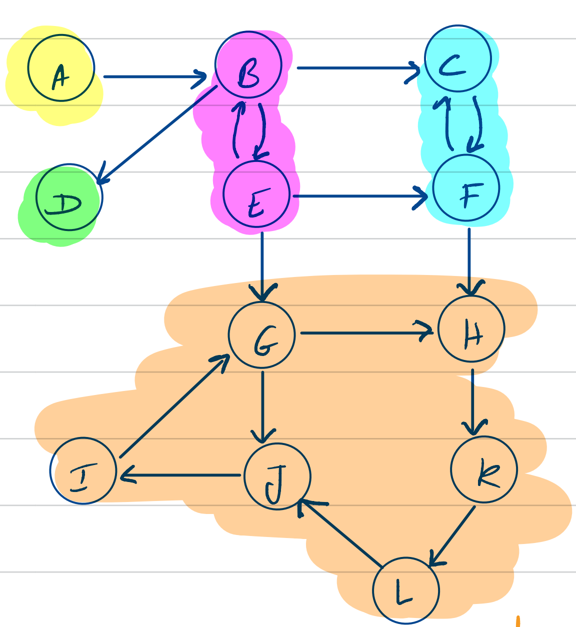
Observação 1:

Observação 2:

Componentes Fortemente Ligados - Grafos Transpostos



Grafo Original



Grafo Transposto

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

Componentes Fortemente Ligados - Algoritmo de Kosaraju-Sharir

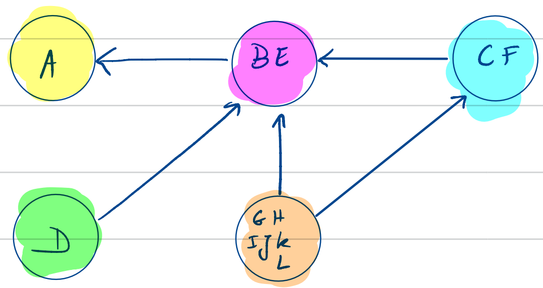
Observação 1: O SCC com maior tempo de fim no grafo G é um SCC source no grafo dos SCCs.

Observação 2: O vértice com maior tempo de fim pertence ao SCC com maior tempo de fim.

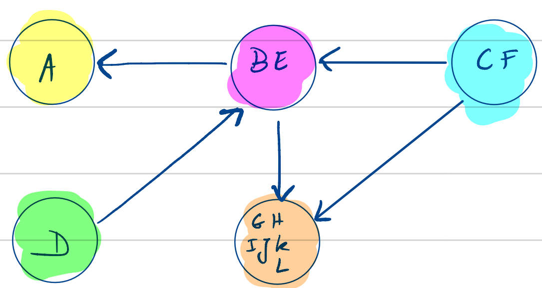
Observação 3: O SCC com maior tempo de fim no grafo G é um SCC sink no grafo dos SCCs do grafo transposto.

Observação Final: O vértice com maior tempo de fim pertence a um SCC sink no grafo dos SCCs do grafo transposto.

Grafo Original



Grafo Transposto

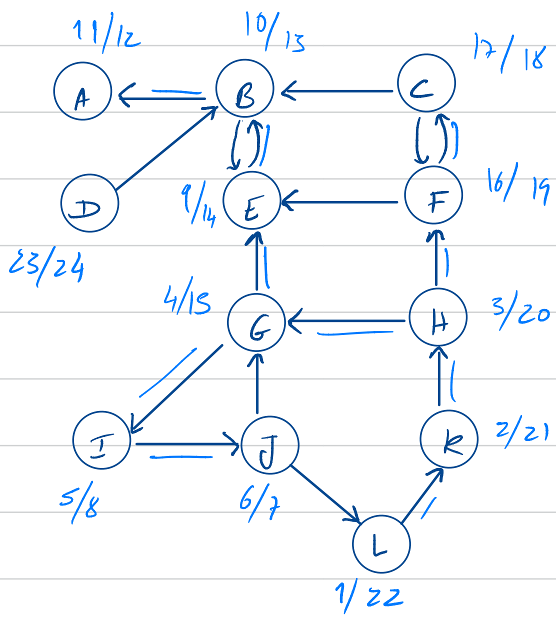


Componentes Fortemente Ligados - Algoritmo de Kosaraju-Sharir

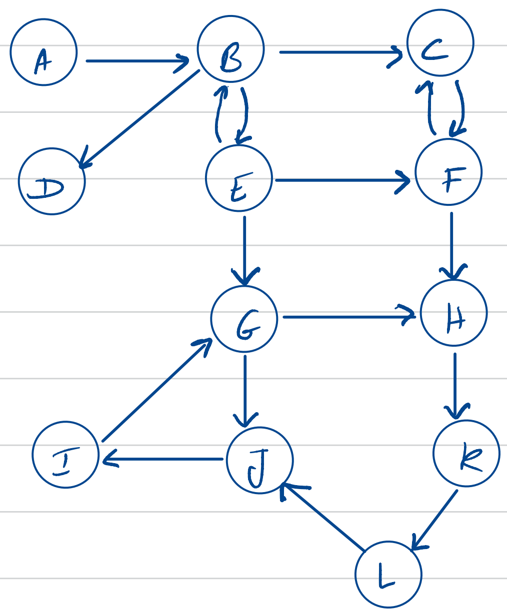
SCCs(G)

- 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 1ª DFS
  - O conjunto dos vértices de cada DFS corresponde a um SCC

1ª DFS:



2ª DFS:

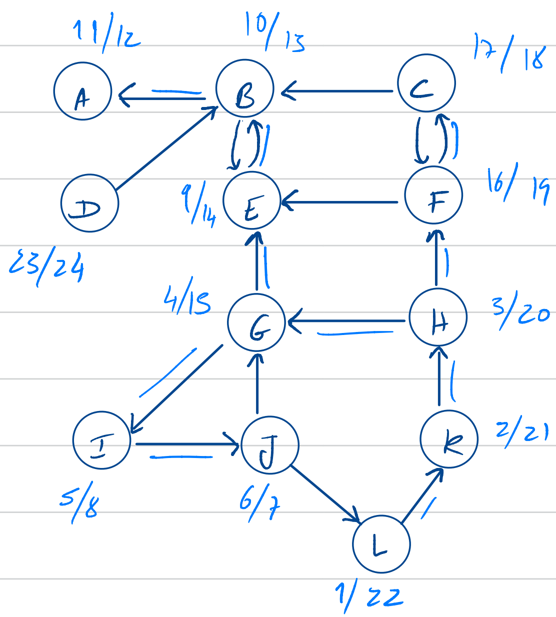


Componentes Fortemente Ligados - Algoritmo de Kosaraju-Sharir

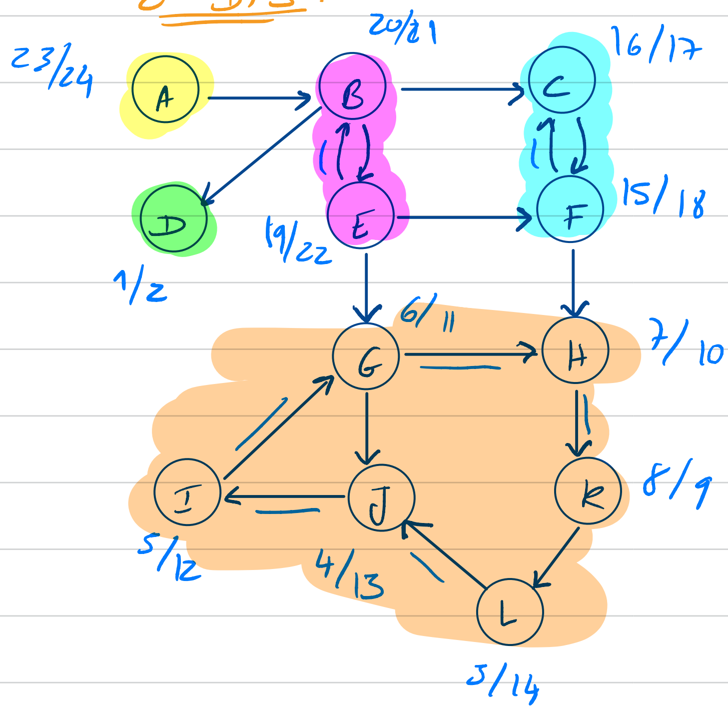
SCCs(G)

- 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 1ª DFS
  - O conjunto dos vértices de cada DFs corresponde a um SCC

1ª DFS:



2ª DFS:



Componentes Fortemente Ligados - Algoritmo de Kosaraju-Sharir

Correcção: Admitimos que as k primeiras árvores encontradas são SCC's e q a $(k+1)$-ésima árvore tb é um SCC.

- Seja x o primeiro vértice encontrado da $(k+1)$-ésima
  Há dois factos a provar:

(I) Todos os vértices do SCC q contém x, C_x, são descendentes de x.

(II) Todos os descendentes de x pertencem a C_x

Componentes Fortemente Ligados - Algoritmo de Kosaraju-Sharir

Correcção: Admitimos que as k primeiras árvores encontradas são SCC's e q a $(k+1)$-ésima árvore tb é um SCC.

- Seja x o primeiro vértice encontrado do $(k+1)$-ésima
  Há dois factos a provar:

(I) Todos os vértices do SCC q contém x, C_x, são descendentes de x.

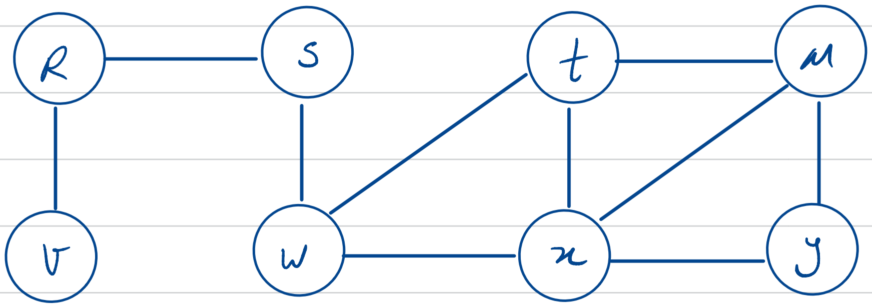
$$y \in C_x \Rightarrow y \text{ é descendente de } x$$

Nota: Fim dos slides da Aula 6

(II) Todos os descendentes de x pertencem a C_x

$$y \text{ é descendente de } x \Rightarrow y \in C_x$$

Procura em Largura Primeiro
(Breadth First Search - BFS)



Definição:

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

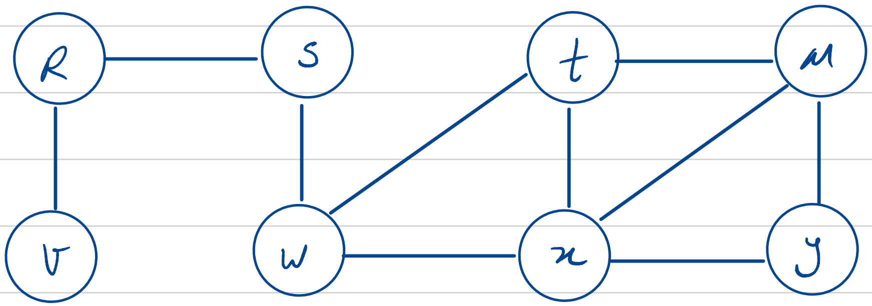
Propriedade [Desigualdade Triangular]

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

Exemplo:

- $\delta(s,y)$?
- $\delta(s,x)$?

Procura em Largura Primeiro
(Breadth First Search - BFS)

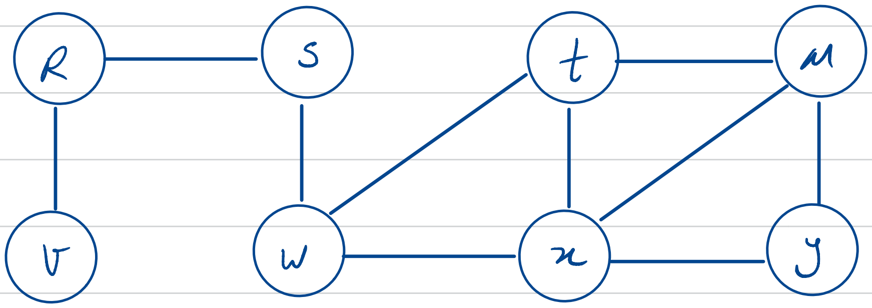


Objectivo:

- Calcular para todos os vértices $v \in V$: $\delta(s, v)$

- Calculamos para cada vértice $u \in V$:
  - $d[u]$: estimativa de $\delta(s, v)$ $\longrightarrow$ No fim do algoritmo $d[v] = \delta(s, v)$
  - $\pi[u]$: pai do vértice u na árvore BFS

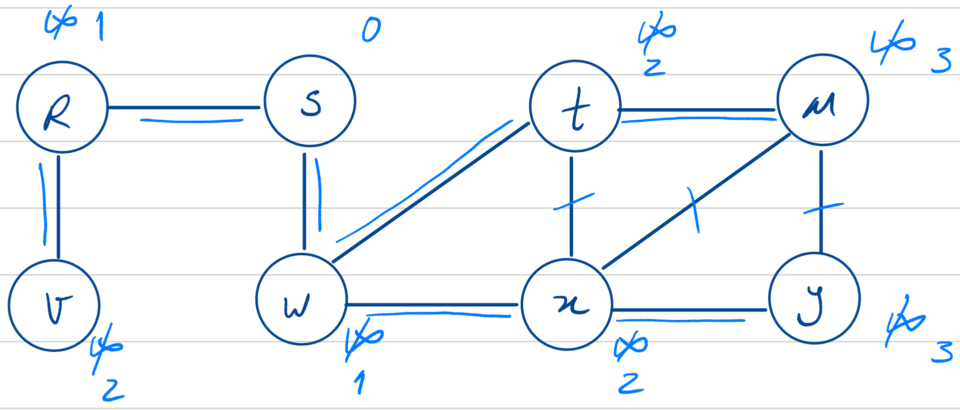
Procura em Largura Primeiro
(Breadth First Search - BFS)



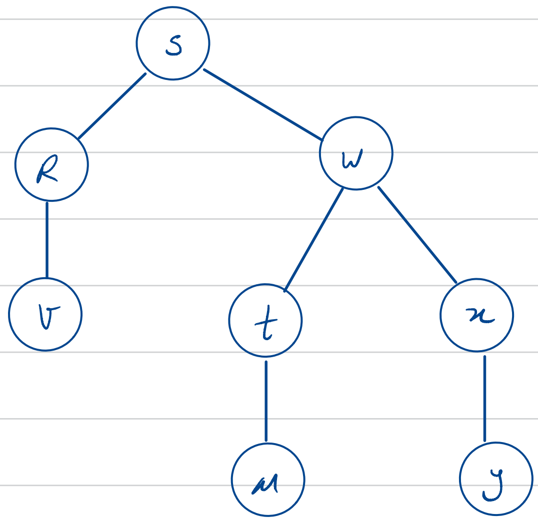
- Calculamos para cada vértice $u \in V$:
  - $d[u]$: estimativa de $\delta(s, v)$
  - $\pi[u]$: pai do vértice u na árvore BFS

- 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



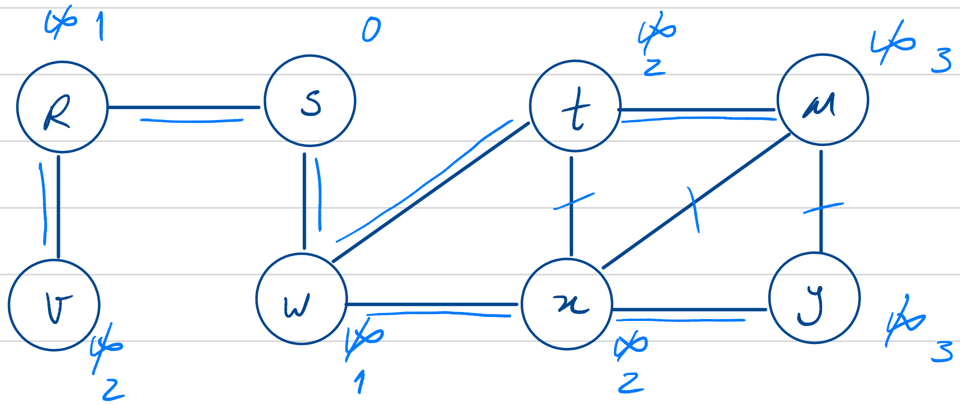
• Calculamos para cada vértice $u \in V$:

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

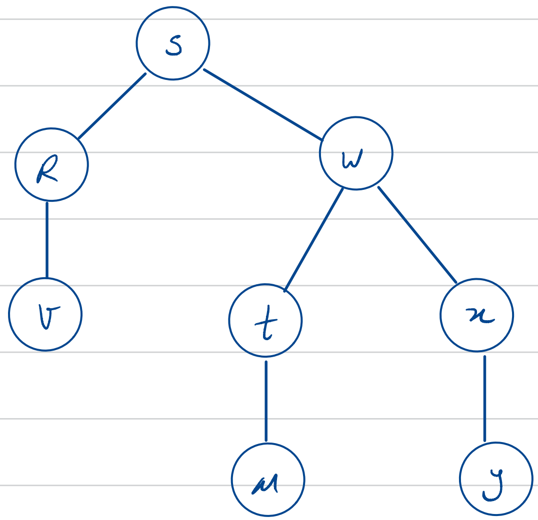
- $d[u]$: estimativa de $\delta(s, v)$

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

Procura em Largura Primeiro
(Breadth First Search - BFS)



Árvore BFS



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

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

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

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

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, colour)

Q := new Queue( ); Q.enqueue(s);

set up da Fila que faz a gestão dos vértices

while (!Q.empty( )) {

v := Q.dequeue( );

}

Loop principal

Procura em Largura Primeiro
(Breadth First Search - BFS)

```
BFS (G, s)
    for each v ∈ G.V \ {s}
        v.π := Nil ; v.color := White ; v.d := +∞ ;
    s.π := Nil ; s.color := Gray ; s.d := 0 ;

    Q := New Queue ( ); Q. enqueue (s) ;

    while ( !Q.empty() ) {
        v := Q. dequeue () ;
        for each w ∈ G.Adj[v]
            if w.color == White
                Q. enqueue ( w )
                w.color := Gray ; w.π := v ; w.d := v.d + 1
        v.color := Black
    }
```

Set up dos vértices (d, π, color)

Set up da Fila que faz a gestão dos vértices

Loop principal

Procura em Largura Primeiro – Complexidade

```
BFS (G, s)
  for each v ∈ G.V \ {s}
      v.π := Nil ; v.color := White ; v.d := +∞ ;
  s.π := Nil ; s.color := Gray ; s.d := 0 ;

  Q := New Queue ( ) ; Q. enqueue (s) ;

  while ( !Q.empty() ) {
      v := Q. dequeue ( ) ;
      for each w ∈ G. Adj [v]
          if w. color == White
              Q. enqueue ( w )
              w. color := Gray ; w.π := v ; w.d := v.d + 1
      v. color := 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 ∈ G.V \ {s}
      v.π := Nil ;  v.color := White ;  v.d := +∞ ;
  s.π := Nil ;  s.color := Gray ;  s.d := 0 ;

  Q := New Queue( ) ;  Q.enqueue(s) ;

  while (!Q.empty( )) {
      v := Q.dequeue( ) ;
      for each w ∈ G.Adj[v]
          if w.color == White
              Q.enqueue(w)
              w.color := Gray ;  w.π := v ;  w.d := v.d + 1
      v.color := 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 $v \in G.V \setminus \{s\}$
    $v.\pi := Nil$; $v.color := White$; $v.d := +\infty$;
  $s.\pi := Nil$; $s.color := Gray$; $s.d := 0$;

  Q := New Queue ( ); Q. enqueue (s) ;

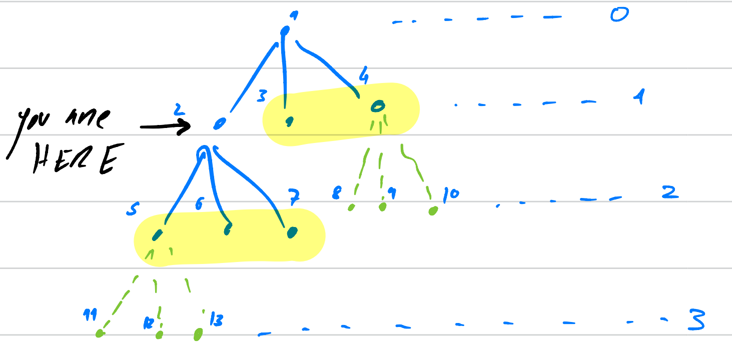
  while (!Q.empty()) {
      v := Q. dequeue ( ) ;
      for each $w \in G.Adj[v]$
        if w. color == White
          Q. enqueue (w)
          $w.color := Gray$; $w.\pi := v$; $w.d := v.d + 1$
      v. color := Black
  } ↳ Loop 2

↳ Loop 1

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

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

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



Provas para estudo individual

Teorema 6.1

O algoritmo de Kosaraju-Sharir é correcto.

Prova:

- Provar a correcção do algoritmo de Kosaraju-Sharir implica mostrar q as árvores da floresta gerada pela 2ª DFS correspondem aos SCCs do grafo original.

- A prova faz-se por indução no nº n de árvores encontradas.
  Caso base $n=0 \Rightarrow$ Não há nada a provar.

- Caso Indutivo: Suponhamos q o algoritmo encontra $(n+1)$ árvores
  - Pela hipótese de indução concluímos q 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 primeira árvore. Temos de mostrar que: T_{n+1} é um SCC.
  - Seja x o 1º vértice encontrado em T_{n+1}. Há que mostrar que: $T_{n+1} = C_x \rightarrow$ componente q contém x

- Mostramos separadamente que:
  - $T_{n+1} \subseteq C_x$
  - $C_x \subseteq T_{n+1}$

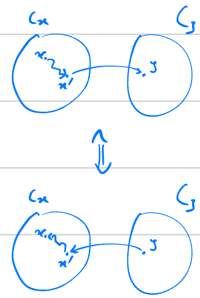
(I) $C_x \subseteq T_{n+1}$

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 Caminho 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 $y \in T_n$ e $y \notin C_x$. (Observemos q y tb não pertence aos componentes até aqui identificados).

Segue q:



em G^T

$\Updownarrow$

em G

$\rightarrow$ Contradiz a hipótese de que visitamos os vértices por ordem decrescente de tempo de fim.

Isto implica que $f(C_y) > f(C_x)$

Teorema 6.2

O Invariante 2 de 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_1 do início da fila e colocamos no fim da fila os seus sucessores brancos.

- Primeiro sucessor branco: u

$u.d = v_1.d + 1$

$Q' = \langle v_2, \dots, v_n, u \rangle$

Há que mostrar que:

(I) $u.d \geq v_n.d$

(II) $u.d \leq v_2.d + 1$

(I) $v_n.d \leq v_1.d + 1 = u.d$ ✓

(II) $u.d \leq v_2.d + 1$

$v_1.d + 1 \leq v_2.d + 1$

$v_1.d \leq v_2.d$ ✓

Teorema 6.3

O Invariante 1 da BFS é mantido pelo algoritmo.

Prova

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

$s.d = 0 = \delta(s,s)$

Passo

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

Soja v o vizinho branco de u:

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

Teorema 6.4 [Correcção BFS]

O algoritmo BFS é correcto.

Prova:

Quando o algoritmo BFS(G,s) termina temos q:

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

- Provamos a correcção do algoritmo por contradição. Suponhamos q existe um nó u tal que depois da execução de BFS(G,s), $u.d \neq \delta(s, u)$.
- Assumimos, sem perda de generalidade, que u é o primeiro vértice no caminho mais curto q o liga a s tal cuja distância calculada não coincide c/ a distância mínima desde por S.
- Seja w o predecessor de u no caminho mais curto que o liga a s, temos q $w.d = \delta(s, w)$.

- Quando w é explorado pela BFS, o vértice u pode ser branco, cinzento, ou preto. Analizamos cada um dos 3 casos separadamente.

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

contradição

[u é preto] O vértice u foi retirado da fila, de onde segue pelo Invariante 2 que:

$$u.d \leq w.d = \delta(s,w) < \delta(s,u)$$

Combinando com o Invariante 1:

$$\delta(s,u) \leq u.d < \delta(s,u)$$ contradição

[u é cinzento] u já se encontra na fila Q para ser explorado.

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

Sabemos pelo Invariante 1 que: $u.d \geq \delta(s,u)$.

Assim concluímos que:

$$\delta(s,u) < u.d \leq \delta(s,u)$$

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