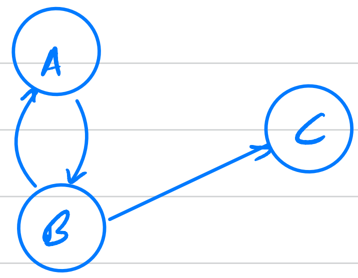
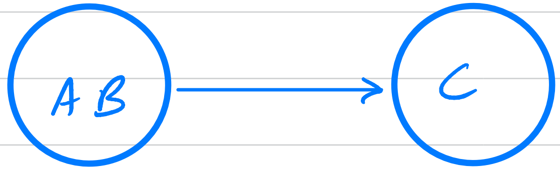
Prática 7

Q1 (CLRS 22.5.3)

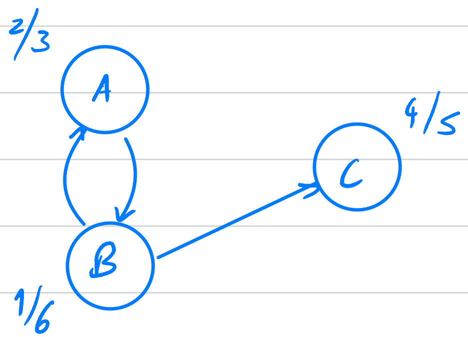
- Algoritmo p/ encontrar SCCs
  - DFS(G)
  - DFS(G) ⇒ percorrendo os vértices por ordem crescente de tempo de fim



- Observação: O SCC com menor tempo de fim é um SCC sink



- Falha: o vértice com menor tempo de fim não pertence necessariamente ao SCC c/ menor tempo de fim.



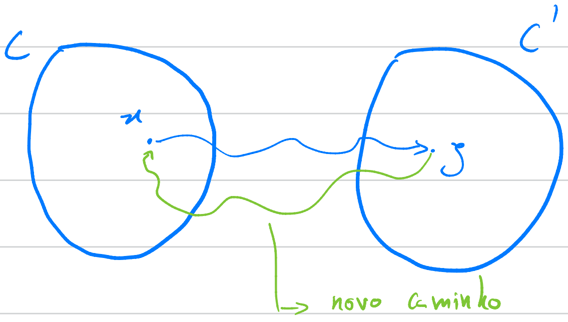
- A 2ª DFS começa pelo vértice A e encontra todo o grafo.

Q2 (CLRS 22.5.1)

- O que acontece ao nº de SCCs de um dado grafo G se adicionarmos um caminho entre dois nós

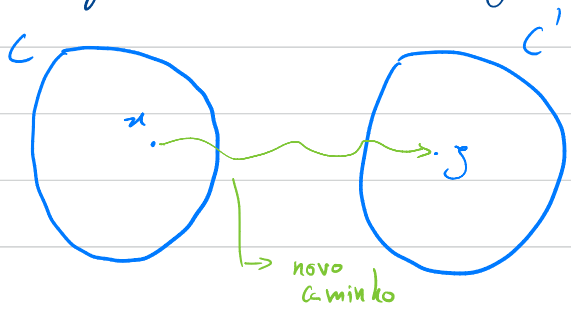
→ Diminui ou mantém-se

• Diminui

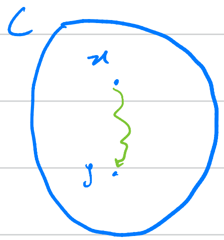


• Mantém-se

Ⓘ Ligamos dois componentes desligados



Ⓘⓘ Nova ligação entre dois vértices dentro do mesmo componente

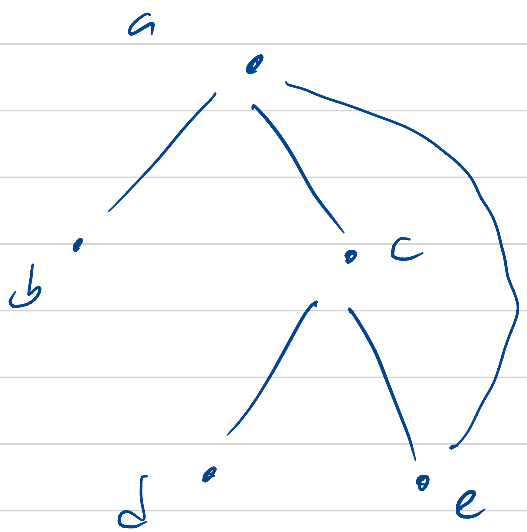


Q3 (CLRS 22.4-3)

- Determinar se um grafo não dirigido contém um ciclo simples em tempo $O(V)$.

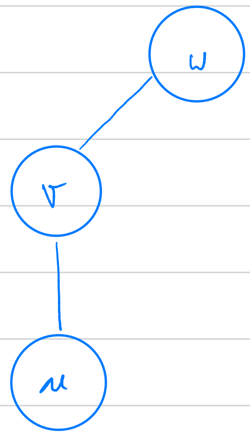
    - DFS modificada

Exemplo



Q3 / (CLRS 22.4-3)

- Determinar se um grafo não dirigido contém um ciclo simples em tempo $O(V)$.



  - DFS modificada

```
DFS(G)
  for v ∈ G.V
      v.visited := false; v.π := Nil
  for v ∈ G.V
      if (!v.visited && DFS-visit(G, v))
          return true
  return false
```

```
DFS_Visit(G, v)
  v.visited := true
  for each u ∈ G.Adj[v]
     if (!u.visited) {
        u.π := v;
        return DFS_Visit(G, u)
     else if (v.π ≠ u) {
        return true
     }
  }
  return false
```

Q3 (CLRS 22.4-3)

- Determinar se um grafo não dirigido contém um ciclo simples em tempo $O(V)$.

  - DFS modificada

```
DFS(G)
  for v ∈ G.V
      v.visited := false; v.π := Nil
  for v ∈ G.V
      if (!v.visited && DFS-visit(G, v))
          return true
  return false
```

```
DFS_Visit(G, v)
   v.visited := true
   for each u ∈ G.Adj[v]
      if (!u.visited) {
          u.π := v;
          return DFS_Visit(G, u)
      else if (v.π ≠ u) {
          return true
      }
   }
   return false
```

Q3 (CLRS 22.4-3)

Determinar se um grafo não dirigido contém um ciclo simples em tempo $O(V)$.

```
DFS(G)
  for v ∈ G.V
    v.visited := false; v.π := Nil
  for v ∈ G.V
    if (!v.visited && DFS-visit(G, v))
      return true
  return false
```

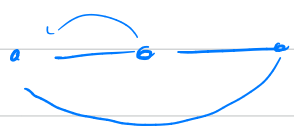
Loop 1

Loop 1
- É executado no máximo $|V|$ vezes (Vértices visitados não são re-visitados)

```
DFS-Visit(G, v)
  v.visited := true
  for each u ∈ G.Adj[v]
    if (!u.visited) {
      u.π := v;
      return DFS-Visit(G, u)
    }
    else if (v.π ≠ u) {
      return true
    }
  }
  return false
```

Loop 2

Loop 2
- O loop 2 é executado no máximo $|E|$ vezes mas um grafo não dirigido acíclico tem no no máximo $|V|-1$ arcos.
- Assim o loop 2 é executado, no pior caso, $2|V|-1$ vezes. O arco $2|V|-1$ é necessariamente arco para trás porque uma DFS num grafo não dirigido só revela arcos p/ trás e arcos da árvore.



Q4 (CLRS 22.5-5)

- Depois de calcular os SCCs calcular o grafo dos SCCs em tempo linear.

- Associar a cada SCC um id único e anotar cada vértice do grafo original com o id do seu SCC. Escrevemos $u.scc$ para denotar o id do SCC a q u pertence.

- $G_{SCC} = (V_{SCC}, E_{SCC})$
  - $V_{SCC} \Rightarrow$ ids dos SCCs de G
  - E_{SCC}

```
Compute ESCC (G, Gscc)
  k := |Gscc.V|    // number of SCCs
  let A[1..k] be a new array whose elements are initially 0
  for each SCC index i in Gscc.V
    let C be the vertices of G in SCC i
    for each u ∈ C
      for each v ∈ G.Adj[u]
        if((u.scc ≠ v.scc) && (A[v.scc] == 0))
          GSCC.addEdge(u.scc, v.scc)
          A[v.scc] := 1
    for each u ∈ C
      for each v ∈ G.Adj[u]
        A[v.scc] := 0
```

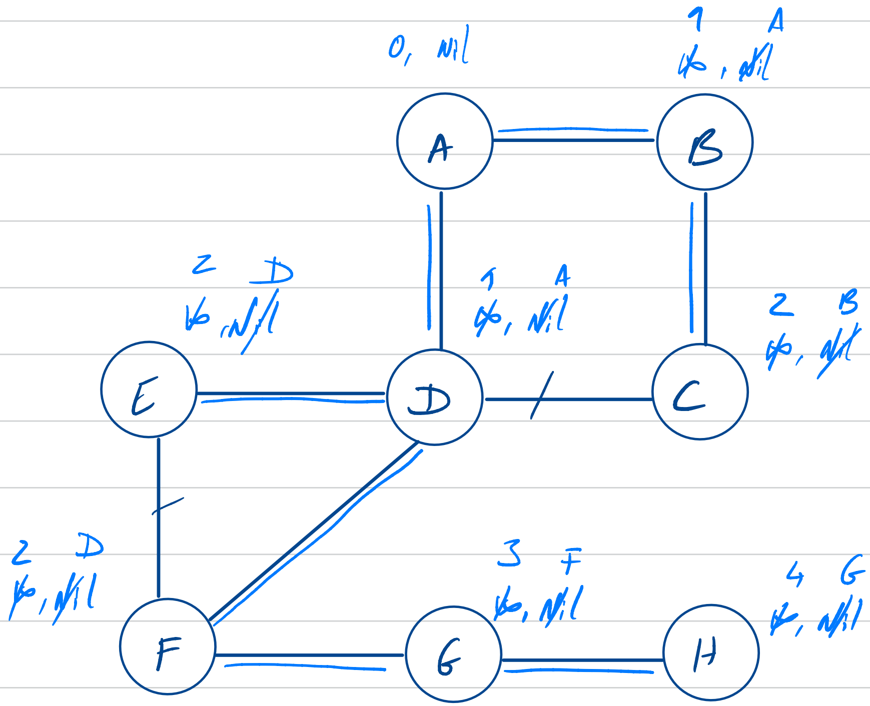
Observação: Visitar as adjacências dos vértices de cada SCC, um SCC de cada vez, mantendo um array A de 0's e 1's de tamanho igual ao nº de SCCs.

$A[j] = 1$ sse já foi encontrado um arco a ligar o SCC q está a ser visitado ao SCC j

Complexidade:

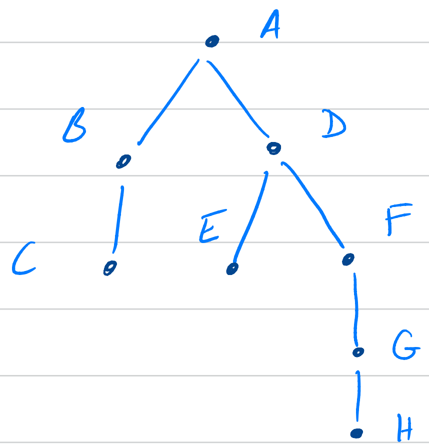
$$\sum_{V_i \in SCC(G)} \left(\sum_{u \in V_i} \left(O(1) + \sum_{v \in Adj[u]} O(1) \right) \right) = O(|V| + |E|)$$

Q5 (TI 06/07 I.1)



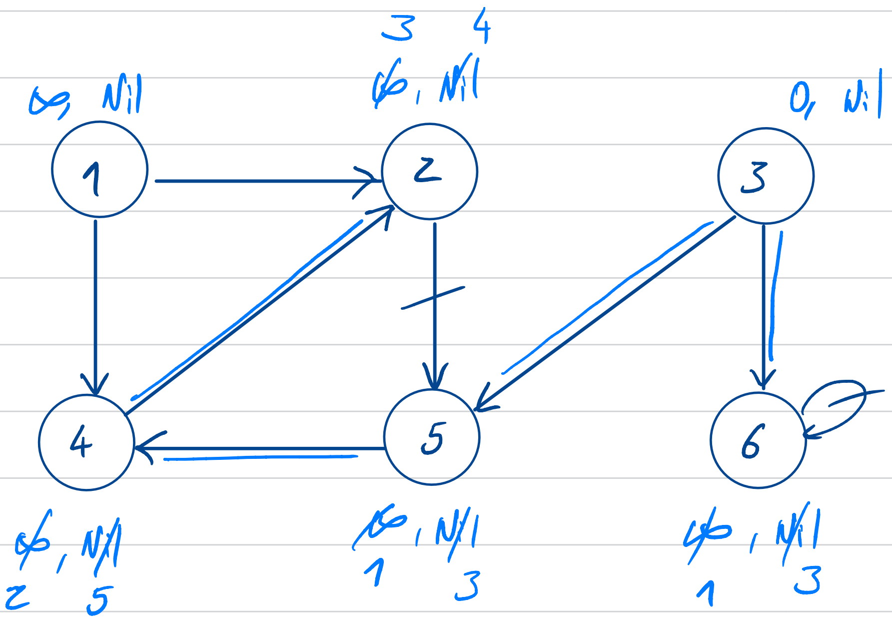
- Queue:

~~A~~ ~~B~~ ~~D~~, ~~C~~, ~~E~~, ~~F~~ ~~G~~, ~~H~~



Anotamos cada nó com o par $(d[u], \pi[u])$

Q6 (CLRS Ex. 22.2-1)



- Começar em 3 e utilizar ordem numérica ...

Q : ~~3~~, ~~5~~, ~~6~~, ~~4~~, ~~2~~

Q7 (CLRS 22.2-7)

Um grafo não dirigido diz-se bipartido se podemos dividir o conjunto V de vértices em dois conjuntos V_1, V_2 tais que:

- $V_1 \cup V_2 = V$
- $V_1 \cap V_2 = \emptyset$
- $out(V_1) \subseteq V_2 \wedge out(V_1) \subseteq V_2$

$out(v) = \{u \mid (v,u) \in G.E\}$

$out(V) = \bigcup_{v \in V} out(v)$

Como determinar se um grafo é bi-partido?

- A ideia é alterar a BFS de forma a associar uma de duas cores a cada vértice quando este é encontrado pela BFS. $O(|V|+|E|)$
- O grafo é bi-partido se não forem encontrados arcos → q ligam vértices da mesma cor. $O(|E|)$

$O(|V|+|E|)$

Q8 RI 08/09 I.3

1) BFS permite identificar os caminhos mais curtos a partir do vértice s.

Correcção DFS:

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

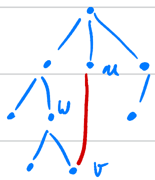
2) Se depois de uma BFS $v.d > v.u + 1$ para dois vértices $u, v \in V$ então $(u,v) \notin E$.

$v.d > v.u + 1$

$\Rightarrow \delta(s,v) > \delta(s,u) + 1$

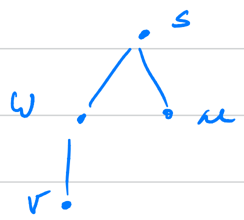
$\Rightarrow (u,v) \notin E$ (desigualdade triangular) T

3) Se o grafo for não dirigido, podem existir arcos para a frente na aplicação da BFS



$\Rightarrow$ Se o arco (u,v), então v seria filho de u e não de w porque os filhos de u são explorados primeiro que os filhos de w. F

4) Sejam u e v dois vértices atingíveis a partir de s tal que $d[v] > d[u]$. Então $d[v] - d[u]$ denota o nº de arcos no caminho mais curto de u para v.



$d[v] = 2$

$d[u] = 1$

$d[v] - d[u] = 1$

Caminho mais curto de u para v:

$\langle u, s, w, v \rangle$

↳ nº de arcos: 3

F

5) Para cada arco $(u, v) \in BF$, $d[v] = d[u] + 1$

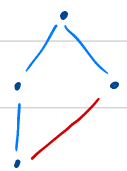
- (u, v) pertence a um caminho mais curto entre s e v:

$$\delta(s, v) = \delta(s, u) + 1$$

$$d[v] = d[u] + 1$$

T

6) Se o grafo for não-dirigido, na aplicação da BFS não existem arcos de cruzamento



F