TÉCNICO LISBOA

Análise e Síntese de Algoritmos

Componentes Fortemente Ligados (SCCs).

CLRS Cap. 22

Prof. Pedro T. Monteiro

IST - Universidade de Lisboa

2024/2025

Contexto

- Revisão [CLRS, Cap.1-13]
  - Fundamentos; notação; exemplos
- Técnicas de Síntese de Algoritmos [CLRS, Cap.15-16]
  - Programação dinâmica [CLRS, Cap.15]
  - Algoritmos greedy [CLRS, Cap.16]
- Algoritmos em Grafos [CLRS, Cap.21-26]
  - Algoritmos elementares
  - Caminhos mais curtos [CLRS, Cap.22,24-25]
  - Árvores abrangentes [CLRS, Cap.23]
  - Fluxos máximos [CLRS, Cap.26]
- Programação Linear [CLRS, Cap.29]
  - Algoritmos e modelação de problemas com restrições lineares
- Tópicos Adicionais
  - Complexidade Computacional [CLRS, Cap.34]

Resumo

Componentes Fortemente Ligados (SCCs)
Algoritmo DFS(G)+DFS(G^T)
Algoritmo Tarjan

Componentes Fortemente Ligados

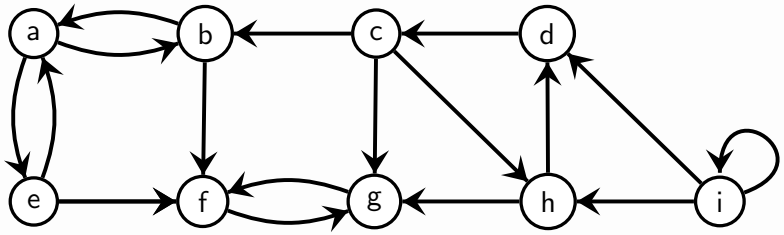
Componente Fortemente Ligado
Dado um grafo dirigido $G = (V, E)$ um componente fortemente ligado (ou Strongly Connected Component – SCC) é um conjunto máximo de vértices $U \subseteq V$, tal que para quaisquer $u, v \in U$, u é atingível a partir de v, e v é atingível a partir de u

Nota: um vértice simples pode definir um SCC

Questão

- Um DAG pode conter SCCs?

Exercício



- Tem ciclos ou é um DAG? Tem ciclos
- Tem quantos SCCs ? 4

Soluções algorítmicas

- Algoritmo baseado na DFS(G)+DFS(G^T) (Kosaraju-Sharir)
- Algoritmo de Tarjan

Grafo Transposto

Dado um grafo dirigido $G = (V, E)$, o grafo transposto de G é definido da seguinte forma:

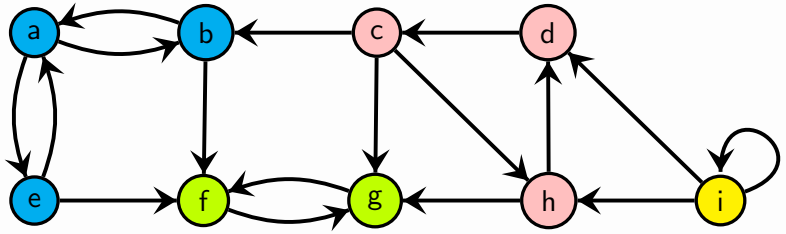
$$G^T = (V, E^T) \text{ tal que : } E^T = \{(v, u) : (u, v) \in E\}$$

Complexidade

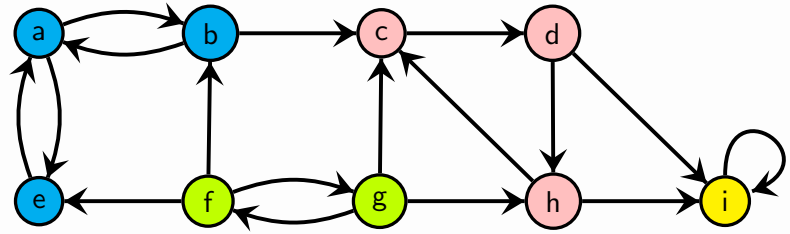
- $O(V + E)$

Observação

G:



G^T:

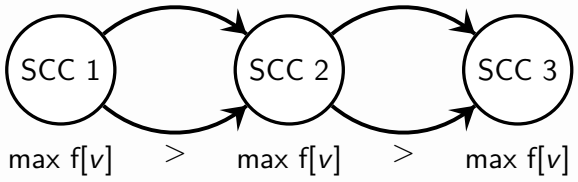


- Os SCCs de G^T são os mesmos de G

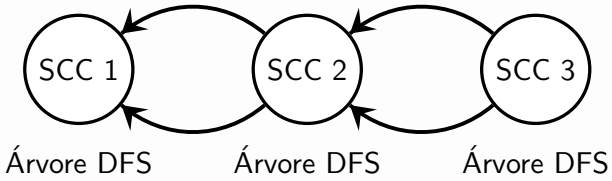
Relembrar: (aula passada)
Num DAG, se existe caminho de u para v, então $f[u] > f[v]$!

Intuição

Em G:



Em G^T:



Algoritmo que utiliza DFS
SCC_DFS(G)

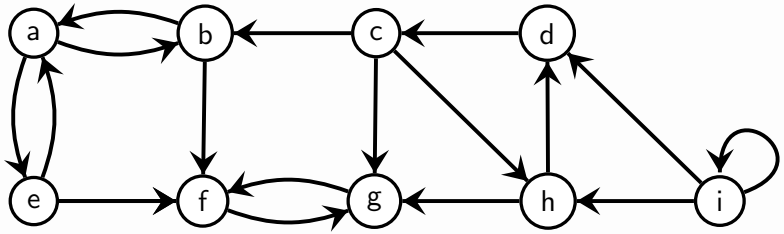
```
1. Executar DFS(G) para cálculo de f[v] (para cada v ∈ G.V)
2. Representar G^T
3. Executar DFS(G^T), considerar f[v] ordem decrescente
return floresta DF
```

(cada árvore DF corresponde a um SCC)

Complexidade

- $\Theta(V + E)$

Exercício: Calcular SCCs do grafo abaixo



Propriedades

- Um grafo de componentes $G^{SCC} = (V^{SCC}, E^{SCC})$ é um DAG
- Se C e C' forem SCCs distintos do grafo G, e existir um caminho de $u \in C$ para $v \in C'$:
  - não pode haver um caminho de v para u
  - $max_{x \in C}\{f[x]\} > max_{y \in C'}\{f[y]\}$
- Segunda DFS visita G^{SCC} seguindo uma ordem topológica

SIAM J. COMPUT.
Vol. 1, No. 2, June 1972

DEPTH-FIRST SEARCH AND LINEAR GRAPH ALGORITHMS*

ROBERT TARJAN†

Abstract. The value of depth-first search or "backtracking" as a technique for solving problems is illustrated by two examples. An improved version of an algorithm for finding the strongly connected components of a directed graph and an algorithm for finding the biconnected components of an undirect graph are presented. The space and time requirements of both algorithms are bounded by $k_1V + k_2E + k_3$ for some constants k_1, k_2, and k_3, where V is the number of vertices and E is the number of edges of the graph being examined.

Key words. Algorithm, backtracking, biconnectivity, connectivity, depth-first, graph, search, spanning tree, strong-connectivity.

Não fará parte do programa da disciplina deste ano!
Ignorar slides daqui para a frente!
(apenas para quem tenha curiosidade)

Algoritmo Tarjan

TÉCNICO LISBOA

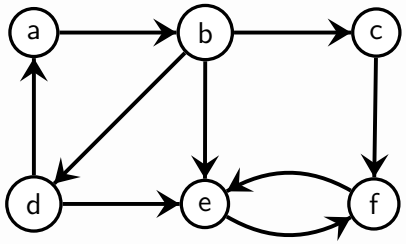
Intuição

- Baseado no algoritmo DFS
- Raíz de um SCC: primeiro vértice do SCC a ser descoberto
- Utilização de arcos para trás e de cruzamento na mesma árvore DF para identificação de ciclos
- $d[v]$: Número de vértices visitados quando v é descoberto
- $low[v]$: O menor valor de $d[]$ atingível por um arco para trás ou de cruzamento na sub-árvore de v
- Se $d[v] = low[v]$, então v é raíz de um SCC

Algoritmo Tarjan

TÉCNICO LISBOA

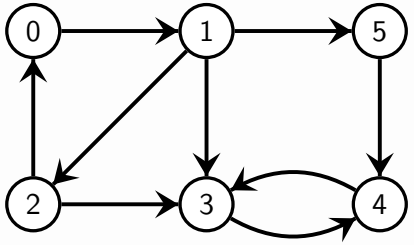
Exemplo valor low[v]



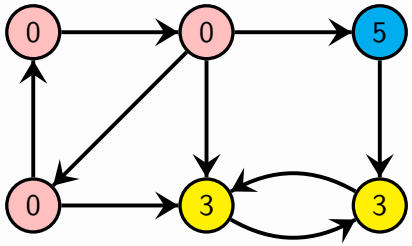
Algoritmo Tarjan

TÉCNICO LISBOA

Exemplo valor low[v]



Exemplo valor low[v]



Invariante de Pilha: conjunto de vértices dos quais é permitido actualizar o valor $low[v]$

Algoritmo de Tarjan

SCC_Tarjan(G)

```
visited ← 0
L ← 0
for u ∈ G.V do
  d[u] ← ∞
end for
for u ∈ G.V do
  if d[u] == ∞ then
    Tarjan_Visit(G, u)
  end if
end for
```

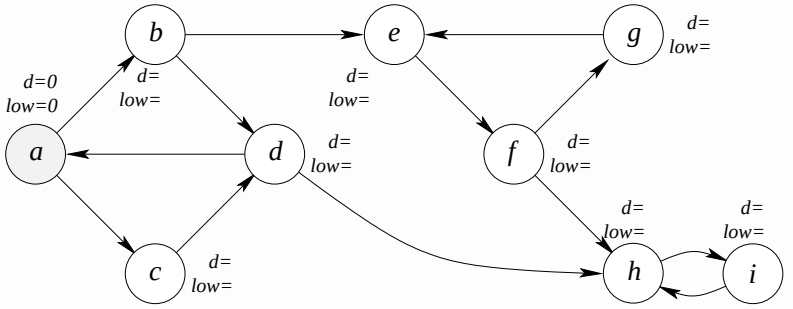
Tarjan_Visit(G,u)

```
d[u] ← low[u] ← visited
visited ← visited + 1
Push(L, u)
for v ∈ G.Adj[u] do
  if d[v] == ∞ or v ∈ L then
    if d[v] == ∞ then
      Tarjan_Visit(G, v)
    end if
    low[u] ← min(low[u], low[v])
  end if
end for
if d[u] == low[u] then
  repeat
    v ← Pop(L)
  until u == v
end if
```


Complexidade

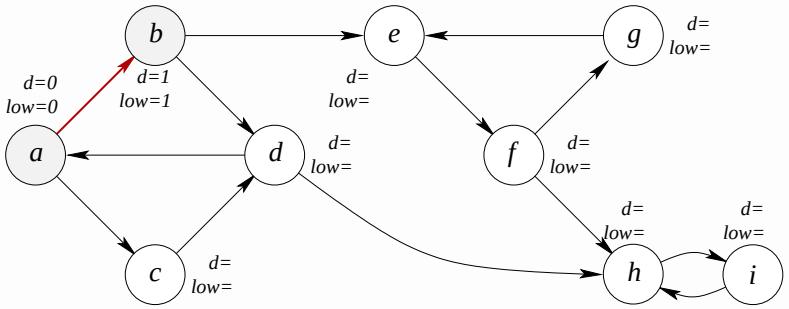
Tempo de execução: $\Theta(V + E)$

- Inicialização: $\Theta(V)$
- Chamadas a Tarjan_Visit: $O(V)$
- Lista de adjacência de cada vértice analisada apenas 1 vez: $\Theta(E)$



L: a *SCCs:*

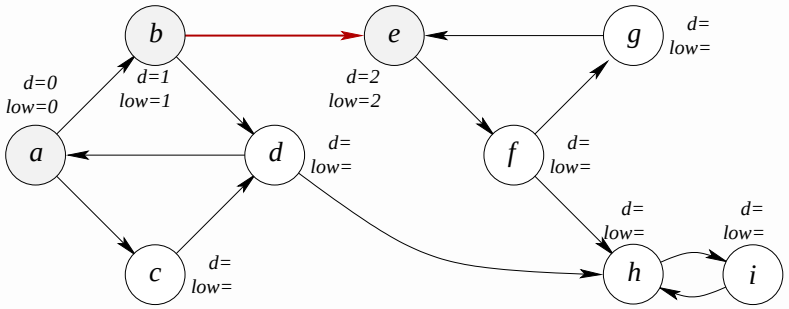
Algoritmo Tarjan



L: a, b

SCCs:

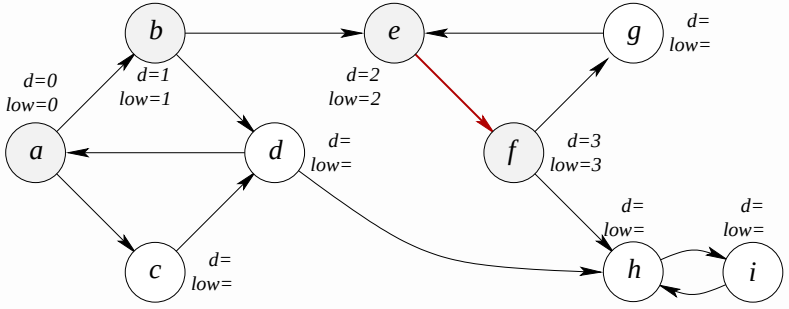
Algoritmo Tarjan



L: a, b, e

SCCs:

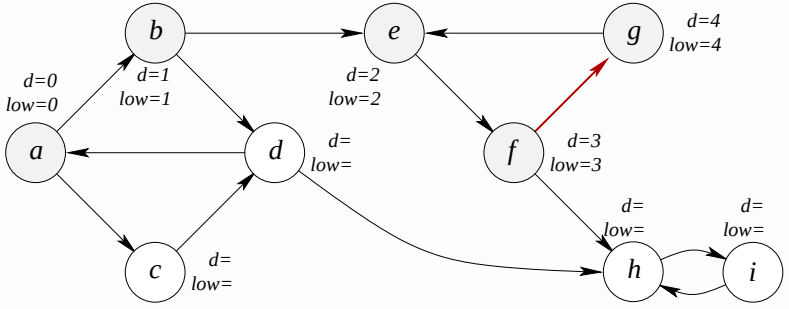
Algoritmo Tarjan



L: a, b, e, f

SCCs:

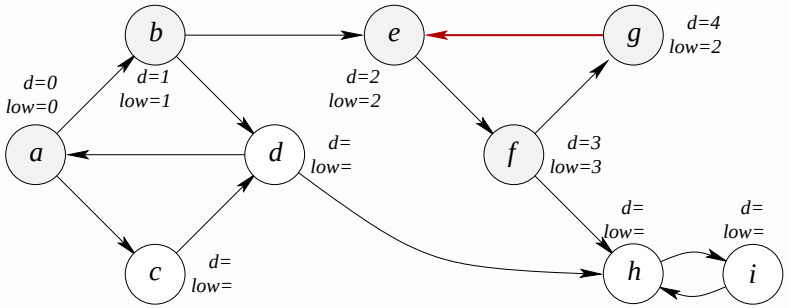
Algoritmo Tarjan



L: a, b, e, f, g

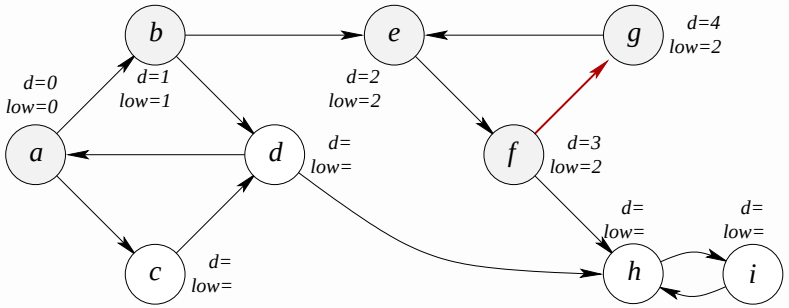
SCCs:

Algoritmo Tarjan



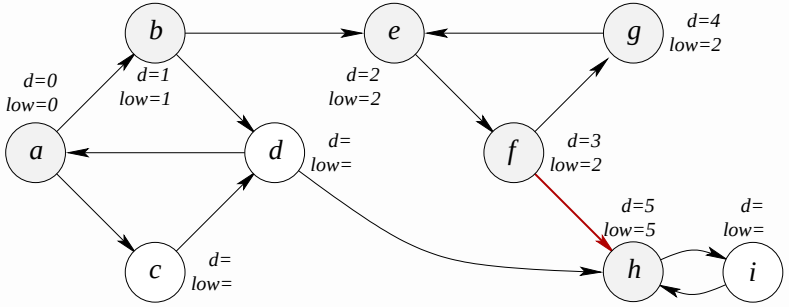
L: a, b, e, f, g *SCCs:*

Algoritmo Tarjan



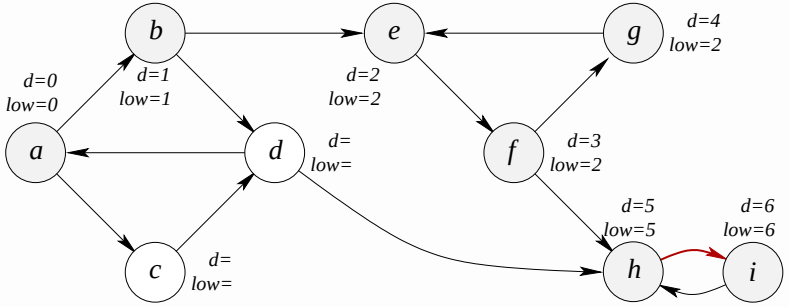
L: a, b, e, f, g *SCCs:*

Algoritmo Tarjan



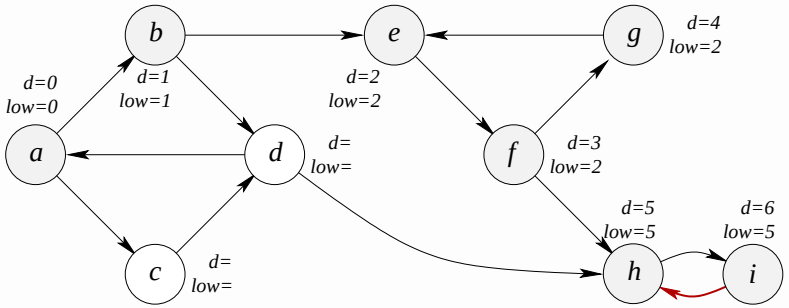
L: a, b, e, f, g, h *SCCs:*

Algoritmo Tarjan



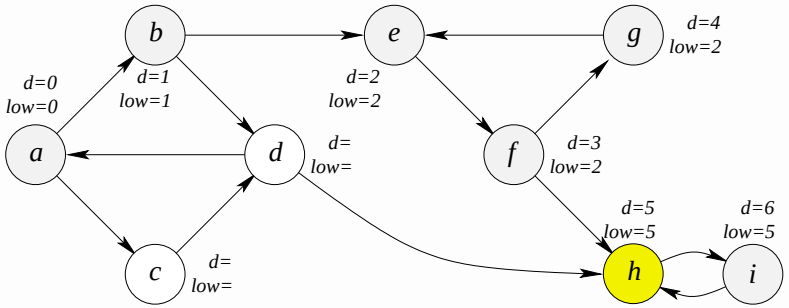
L: a, b, e, f, g, h, i *SCCs:*

Algoritmo Tarjan



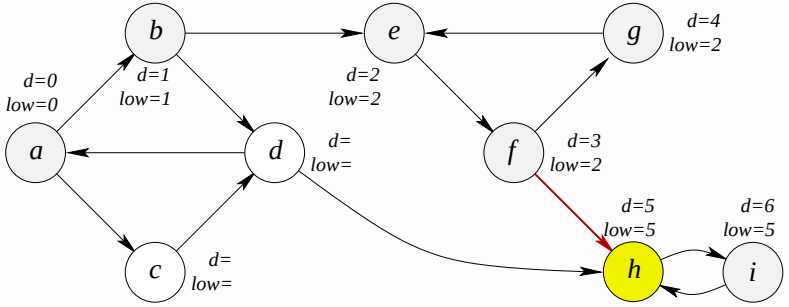
L: a, b, e, f, g, h, i *SCCs:*

Algoritmo Tarjan



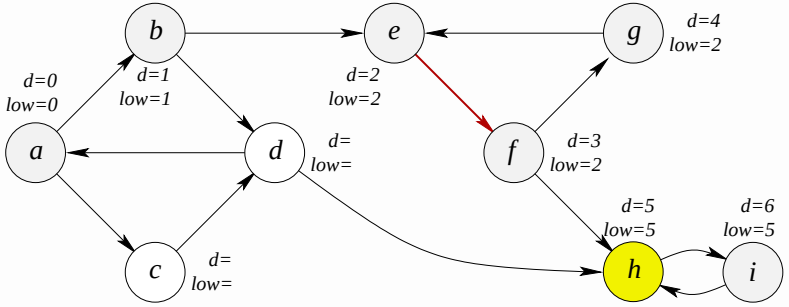
L: a, b, e, f, g *SCCs: {h, i}*

Algoritmo Tarjan



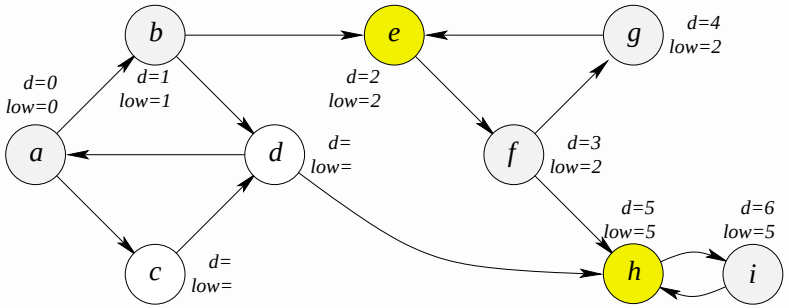
L: a, b, e, f, g *SCCs: {h, i}*

Algoritmo Tarjan



L: a, b, e, f, g *SCCs: {h, i}*

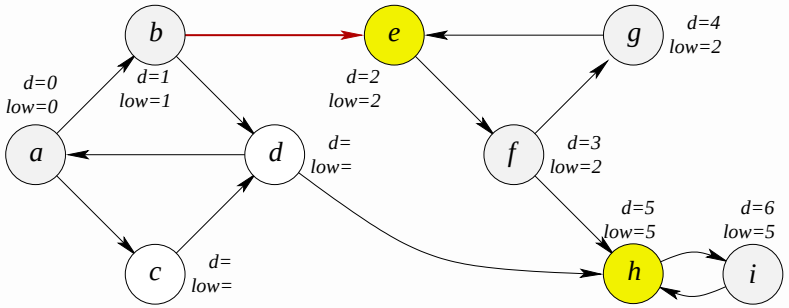
Algoritmo Tarjan



L: a, b

SCCs: {h, i} {e, f, g}

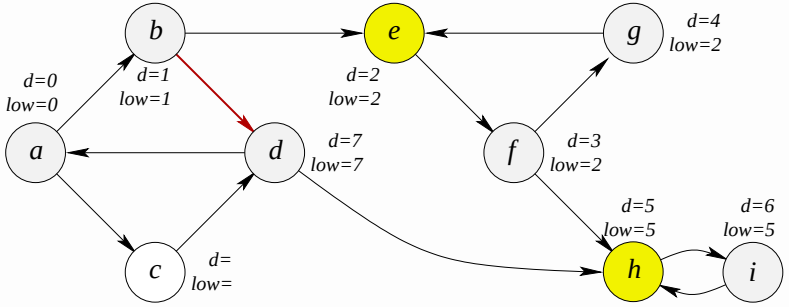
Algoritmo Tarjan



L: a, b

SCCs: {h, i} {e, f, g}

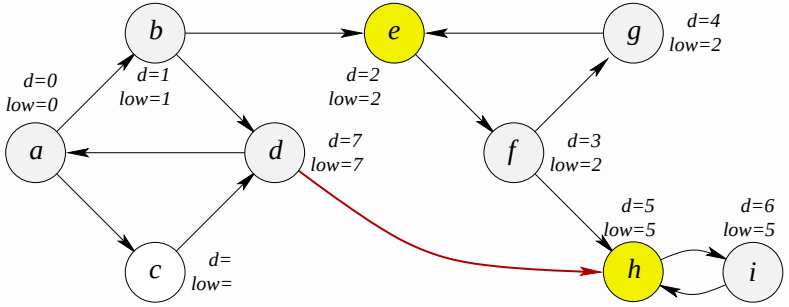
Algoritmo Tarjan



L: a, b, d

SCCs: {h, i} {e, f, g}

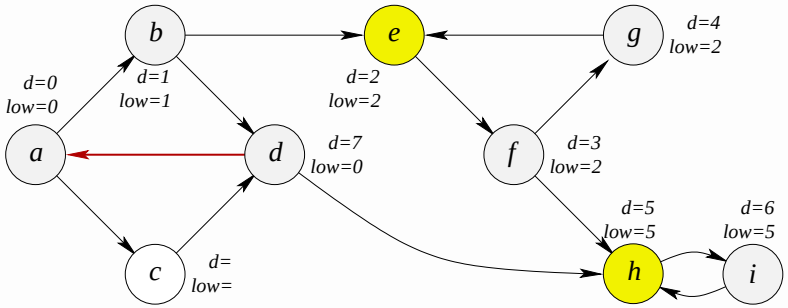
Algoritmo Tarjan



L: a, b, d

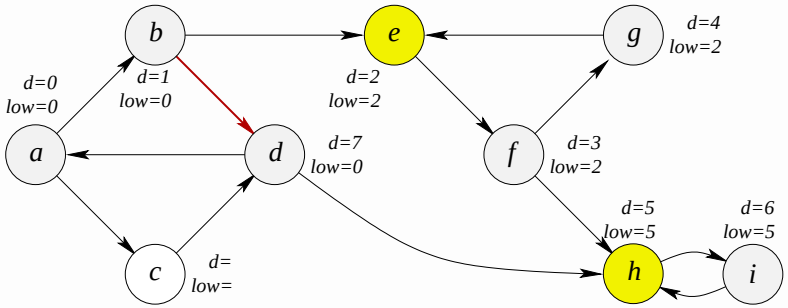
SCCs: {h, i} {e, f, g}

Algoritmo Tarjan



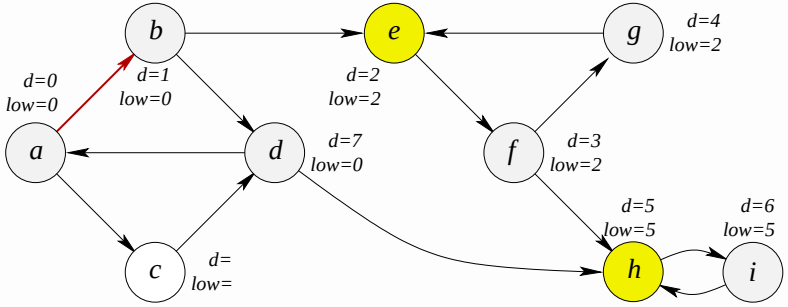
L: a, b, d *SCCs: {h, i} {e, f, g}*

Algoritmo Tarjan



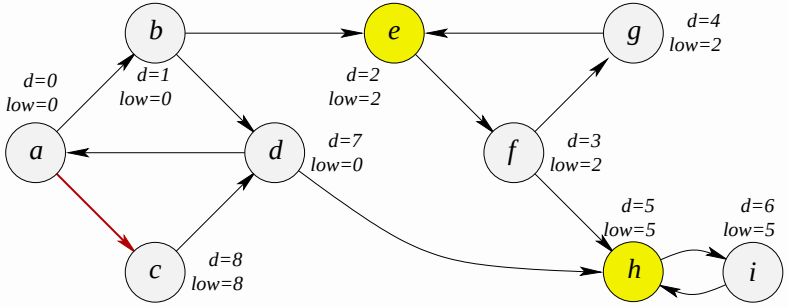
L: a, b, d *SCCs: {h, i} {e, f, g}*

Algoritmo Tarjan

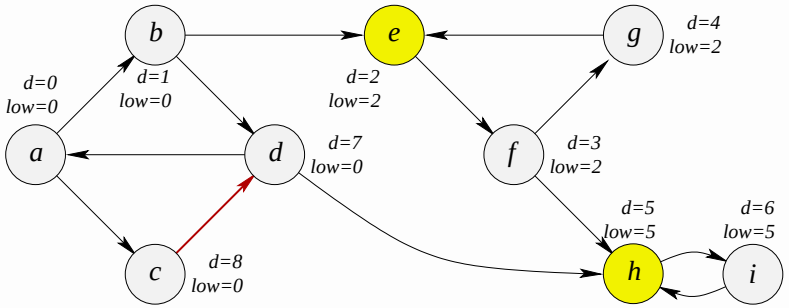


L: a, b, d *SCCs: {h, i} {e, f, g}*

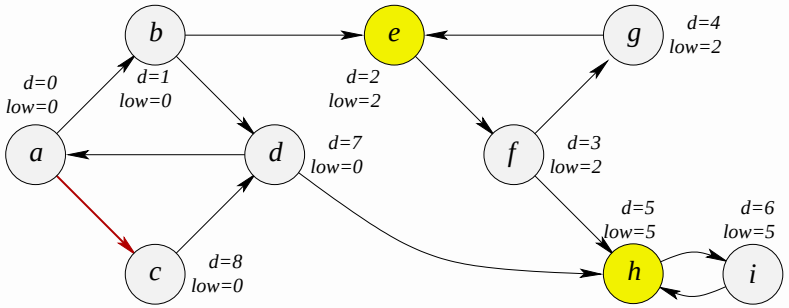
Algoritmo Tarjan



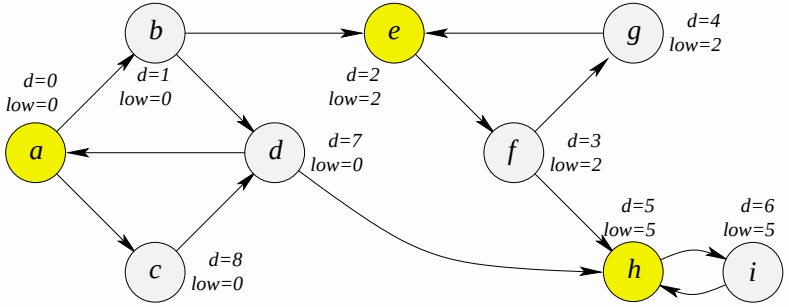
L: a, b, d, c *SCCs: {h, i} {e, f, g}*



L: a, b, d, c *SCCs: {h, i} {e, f, g}*

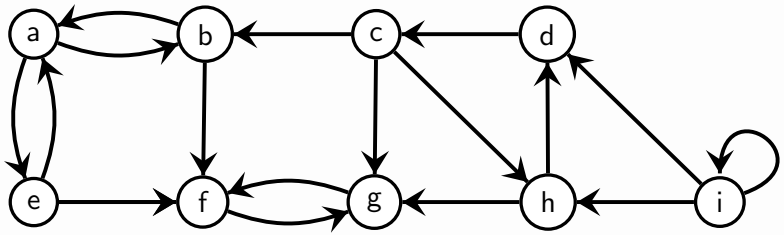


L: a, b, d, c *SCCs: {h, i} {e, f, g}*



L: *SCCs: {h, i} {e, f, g} {a, b, c, d}*

Exercício: Calcular SCCs do grafo abaixo



Resultado secundário

O Algoritmo de Tarjan adicionalmente indica uma ordem topológica entre os SCCs descobertos

- por ordem crescente do valor *low* das raízes dos SCCs
- por ordem inversa da apresentação dos SCCs

Dúvidas?