Análise e Síntese de Algoritmos

Algoritmos Elementares em Grafos [CLRS, Cap. 22]

2011/2012

Contexto

- Revisão [CLRS, Cap.1-13]
  - Fundamentos; notação; exemplos
- Algoritmos em Grafos [CLRS, Cap.21-26]
  - Algoritmos elementares
  - Árvores abrangentes
  - Caminhos mais curtos
  - Fluxos máximos
- Programação Linear [CLRS, Cap.29]
  - Algoritmos e modelação de problemas com restrições lineares
- Técnicas de Síntese de Algoritmos [CLRS, Cap.15-16]
  - Programação dinâmica
  - Algoritmos greedy
- Tópicos Adicionais [CLRS, Cap.32-35]
  - Emparelhamento de Cadeias de Caracteres
  - Complexidade Computacional
  - Algoritmos de Aproximação

Resumo Aula Anterior

Representação de grafos

- Listas de adjacências
- Matrizes de adjacências

Algoritmos Elementares

- Procura em Largura Primeiro (BFS)
  - Caminhos mais curtos no número de arcos
- Procura em Profundidade Primeiro (DFS)
  - Tempos de início (d[]) e de fim (f[])
  - Classificação de arcos

Resumo

Caminhos

Caminhos em Grafos

Dado um grafo $G = (V, E)$, um caminho p é uma sequência $< v_0, v_1, ..., v_k >$ tal que para todo o i, $0 \leq i \leq k - 1$, $(v_i, v_{i+1}) \in E$

- Se existe um caminho p de u para v,então v diz-se atingível a partir de u usando p
- Um ciclo num grafo $G = (V, E)$ é um caminho $< v_0, v_1, ..., v_k >$, tal que $v_0 = v_k$
- Um grafo dirigido $G = (V, E)$ diz-se acíclico (ou Directed Acyclic Graph (DAG)) se não tem ciclos.

Ordenação Topológica

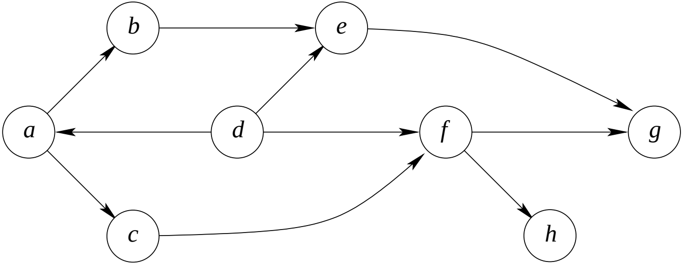
Ordenação Topológica

Uma ordenação topológica de um DAG $G = (V, E)$ é uma ordenação de todos os vértices tal que se $(u, v) \in E$ então u aparece antes de v na ordenação

Soluções Algoritmicas

- Eliminação de vértices
- Utilizando informação de DFS

Ordenação Topológica



Ordenação?

Ordenação Topológica

Pseudo-Código Algoritmo Eliminação de Vértices

Topological-Sort-1(G)

$L = \emptyset$ $\triangleright$ Lista de vértices
$Q = \emptyset$ $\triangleright$ Fila de vértices (FIFO)
for each $v \in G$ $\triangleright$ Inicialização $O(V)$
  do if v sem arcos de entrada (w, v)
    then Enqueue(Q, v)
while $Q \neq \emptyset$ $\triangleright$ Ciclo Principal $O(V + E)$
  do u = Head(Q)
    Eliminar todos os arcos (u, v)
    if v sem arcos de entrada (w, v)
      then Enqueue(Q, v)
    Dequeue(Q)
    Colocar u no fim da lista L
return L

Ordenação Topológica

Pseudo-Código Algoritmo Utilizando informação de DFS

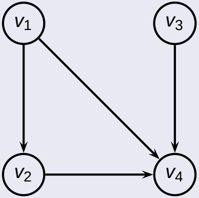
Topological-Sort-2(G)

1 Executar DFS(G) para cálculo do tempo de fim $f[v]$ para cada vértice v
2 Quando um vértice é terminado, inserir no princípio de lista ligada
3 **return** lista ligada de vértices

Complexidade: $O(V + E)$

Ordenação Topológica

Intuição

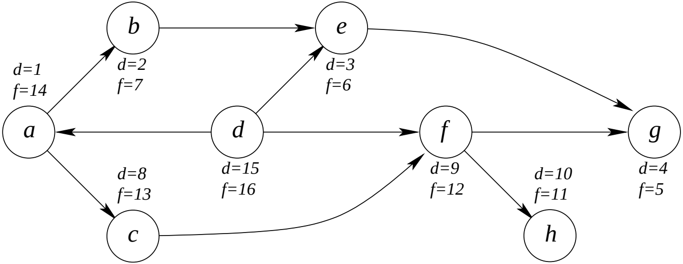


Na DFS:

- Tempo de fim de v_3 é sempre > Tempo de fim de v_4
- Tempo de fim de v_2 é sempre > Tempo de fim de v_4
- Tempo de fim de v_1 é sempre > Tempo de fim de v_2,v_4

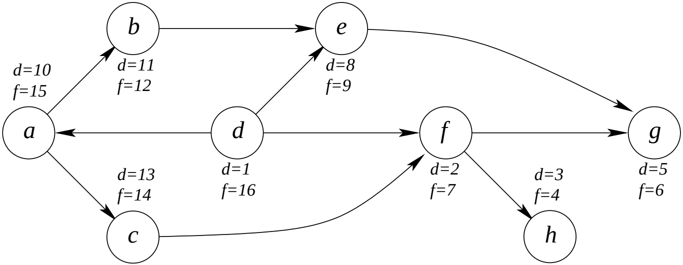
Como o grafo é um DAG, se existe caminho de u para v, verifica-se sempre $f[u] > f[v]$!
Logo, basta ordenar os vértices de forma decrescente dos tempos de fim.

Ordenação Topológica



Ordenação: d, a, c, f, h, b, e, g

Ordenação Topológica



Ordenação: d, a, c, b, e, f, g, h

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.

Grafo Transposto

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

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

Componentes Fortemente Ligados

Pseudo-Código

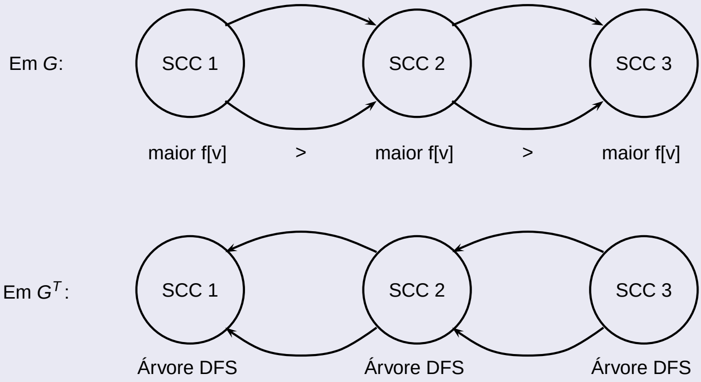
SCCs(G)

```
Executar DFS(G) para cálculo do tempo de fim f[v] para cada vértice v
Representar G^T
Executar DFS(G^T)      ▷ (no ciclo principal de DFS considere os vértices por
                       ▷ ordem decrescente de tempo de fim de DFS(G))
O conjunto de vértices de cada árvore DF corresponde a um SCC
```

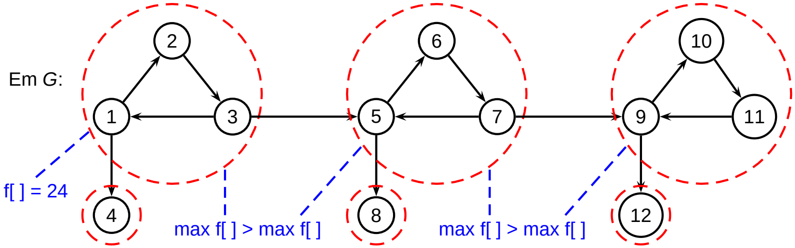
Complexidade: $O(V + E)$

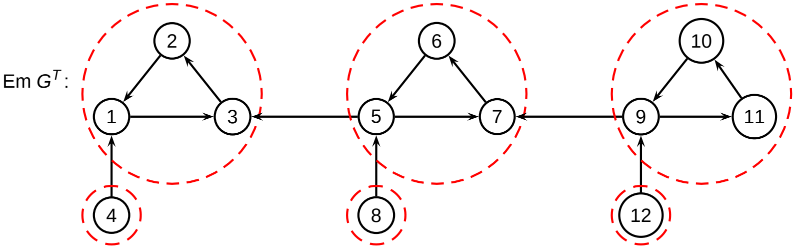
Componentes Fortemente Ligados

Intuição



Componentes Fortemente Ligados





Algoritmo Tarjan

Intuição

- Baseado no algoritmo DFS
- Raíz de um SCC: primeiro vértive 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

Pseudo-Código Algoritmo de Tarjan

Algoritmo de Tarjan

```
SCC_Tarjan(G)
visited = 0
L = ∅
for each vertex u ∈ V[G]
      do d[u] = ∞
for each vertex u ∈ V[G]
      do if d[u] = ∞
            then Tarjan_Visit(u)
```

Algoritmo de Tarjan

```
Tarjan_Visit(u)
d[u] = low[u] = visited
visited = visited + 1
Push(L, u);
for each v ∈ Adj[u]
      do if (d[v] = ∞ || v ∈ L)
            ▷ Ignora vértices de SCCs já identificados
            then if d[v] = ∞
                  then Tarjan_Visit(v)
           low[u] = min(low[u], low[v])
if d[u] = low[u]                              ▷ Raiz do SCC
   then repeat
              v = Pop(L)
              ▷ Vértices retirados definem SCC
        until u = v
```

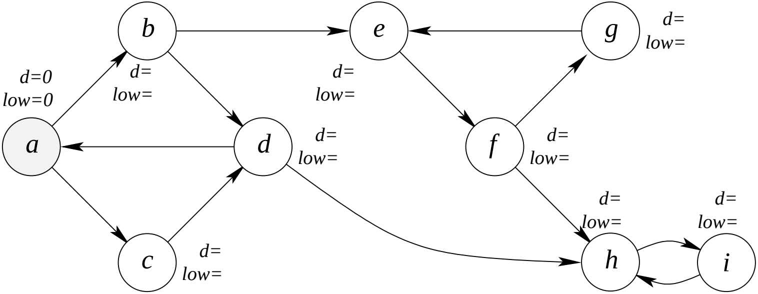
Algoritmo Tarjan

Complexidade

Tempo de execução: $O(V + E)$

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

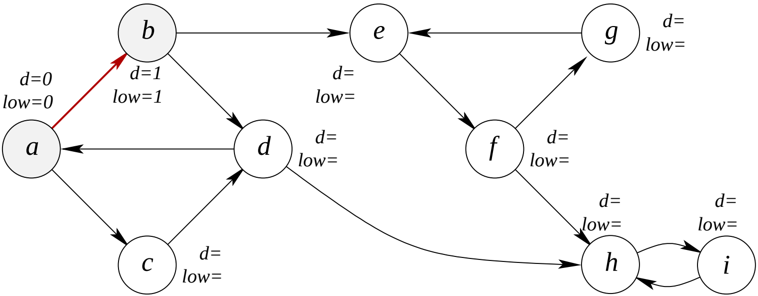
Algoritmo Tarjan



L: a

SCCs:

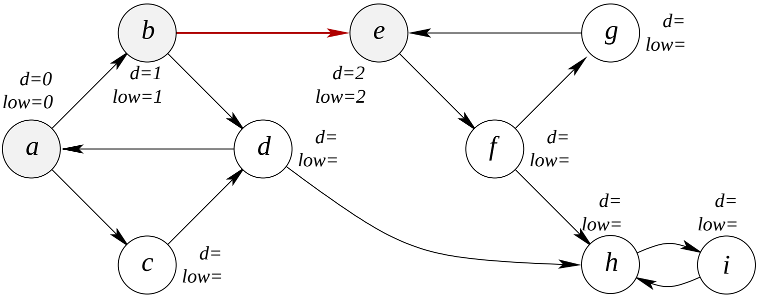
Algoritmo Tarjan



L: a, b

SCCs:

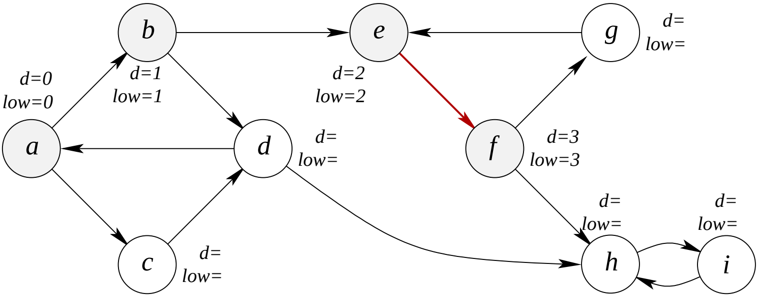
Algoritmo Tarjan



L: a, b, e

SCCs:

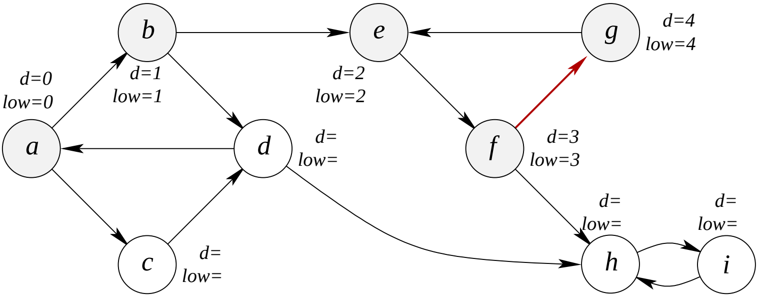
Algoritmo Tarjan



L: a, b, e, f

SCCs:

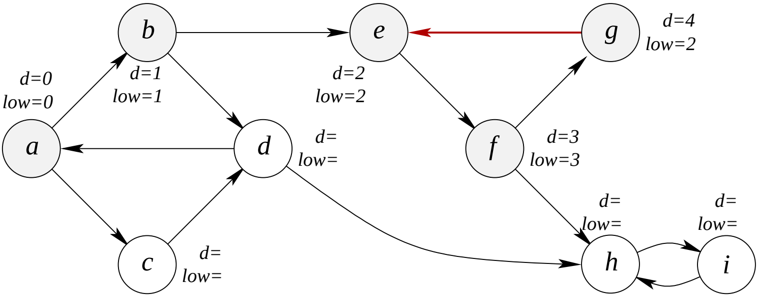
Algoritmo Tarjan



L: a, b, e, f, g

SCCs:

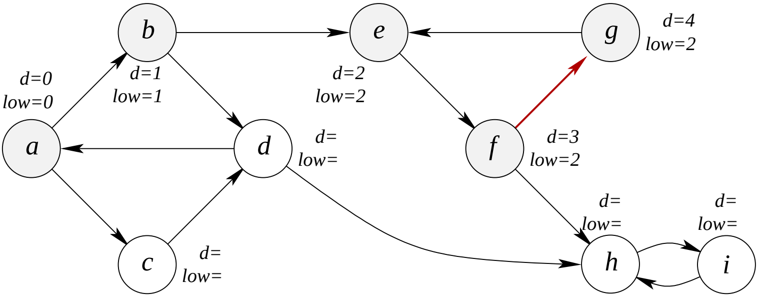
Algoritmo Tarjan



L: a, b, e, f, g

SCCs:

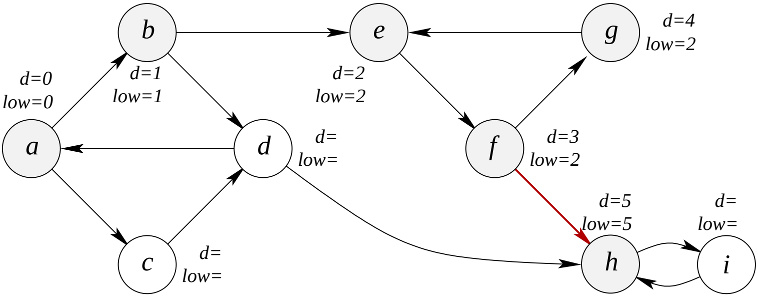
Algoritmo Tarjan



L: a, b, e, f, g

SCCs:

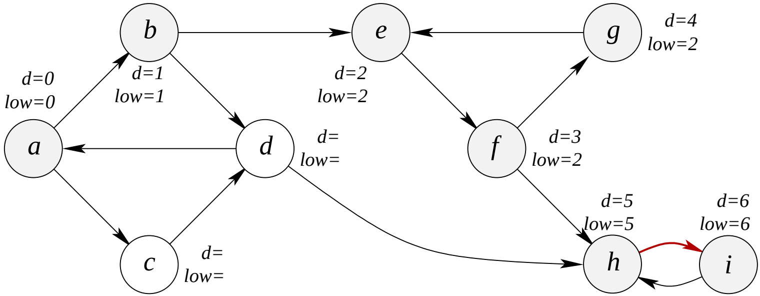
Algoritmo Tarjan



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

SCCs:

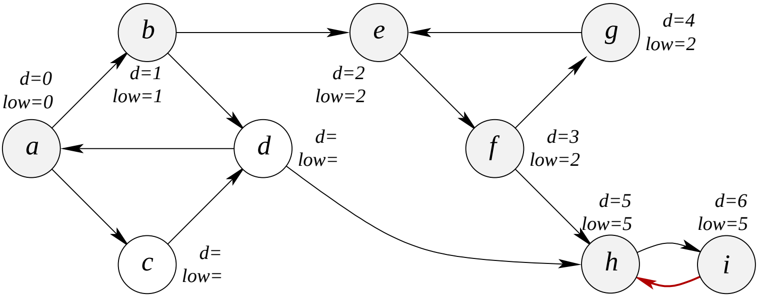
Algoritmo Tarjan



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

SCCs:

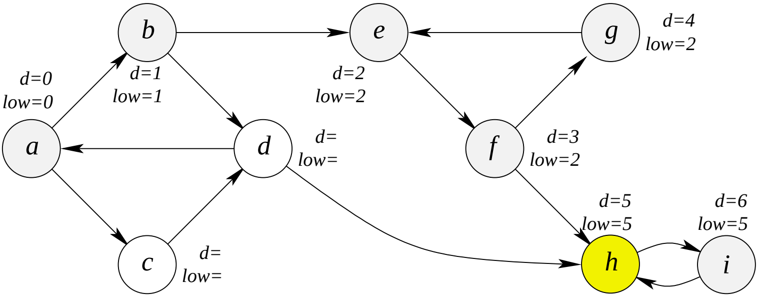
Algoritmo Tarjan



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

SCCs:

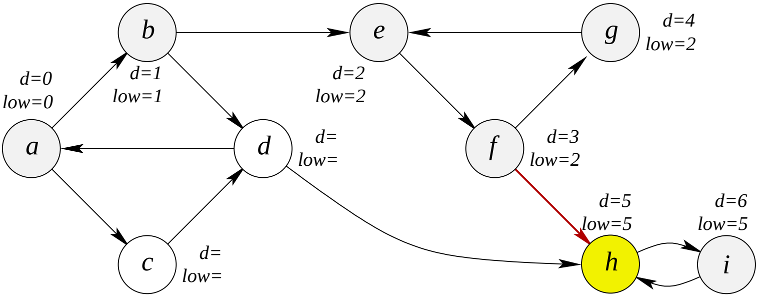
Algoritmo Tarjan



L: a, b, e, f, g

SCCs: {h, i}

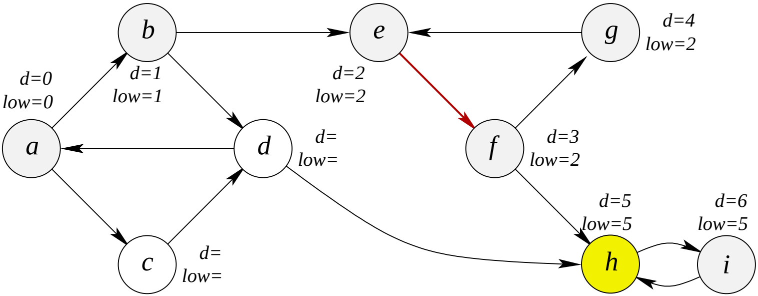
Algoritmo Tarjan



L: a, b, e, f, g

SCCs: {h, i}

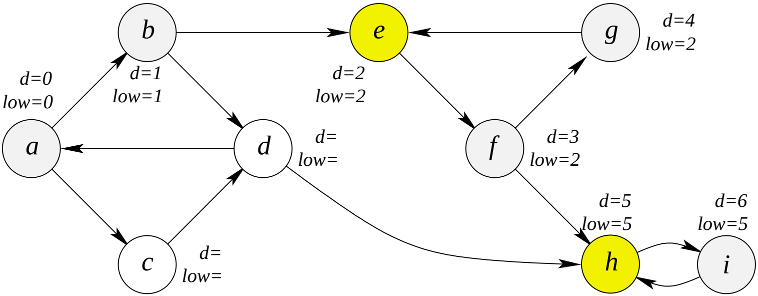
Algoritmo Tarjan



L: a, b, e, f, g

SCCs: {h, i}

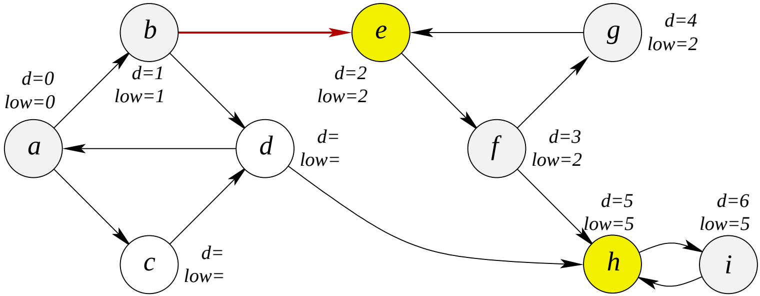
Algoritmo Tarjan



L: a, b

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

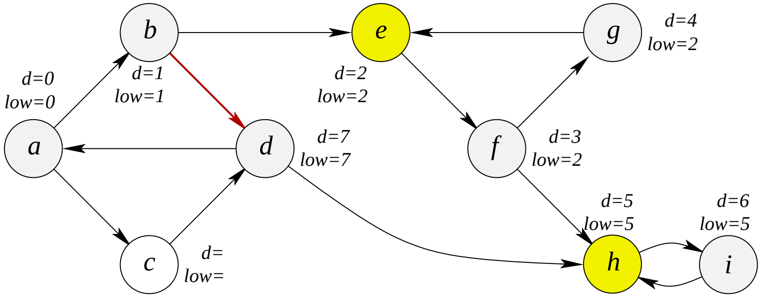
Algoritmo Tarjan



L: a, b

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

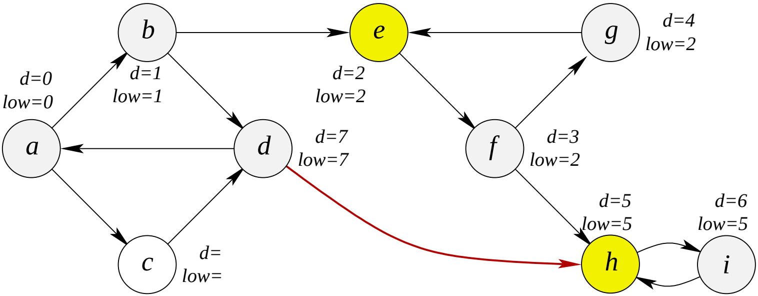
Algoritmo Tarjan



L: a, b, d

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

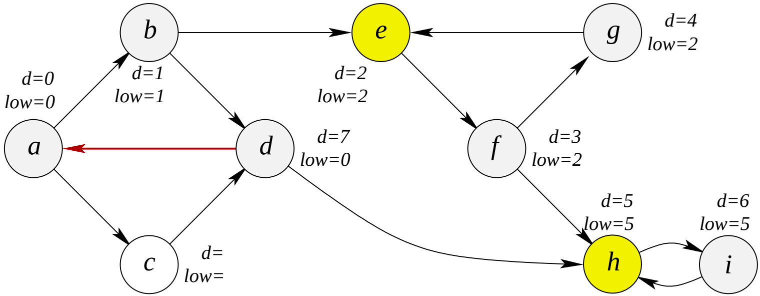
Algoritmo Tarjan



L: a, b, d

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

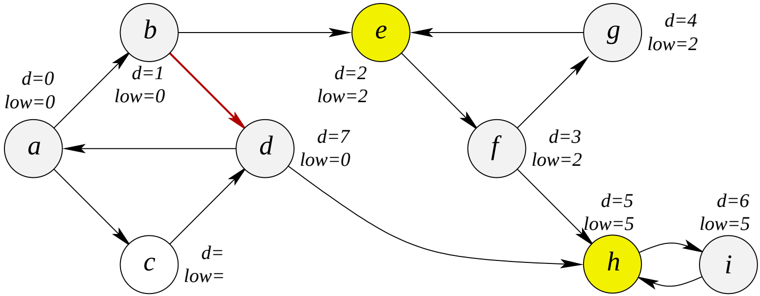
Algoritmo Tarjan



L: a, b, d

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

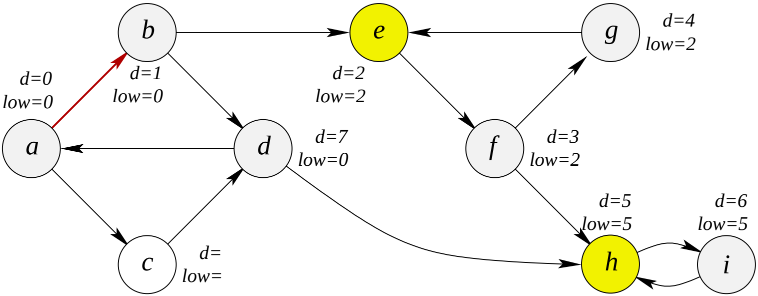
Algoritmo Tarjan



L: a, b, d

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

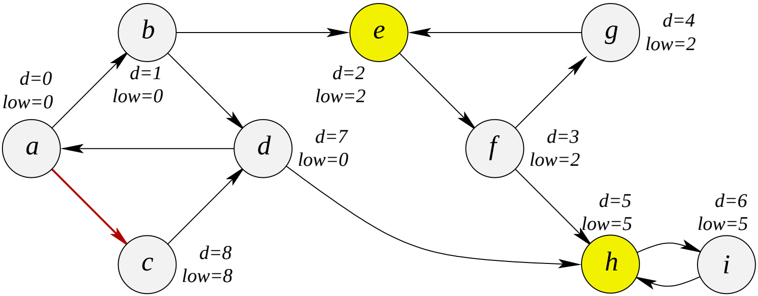
Algoritmo Tarjan



L: a, b, d

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

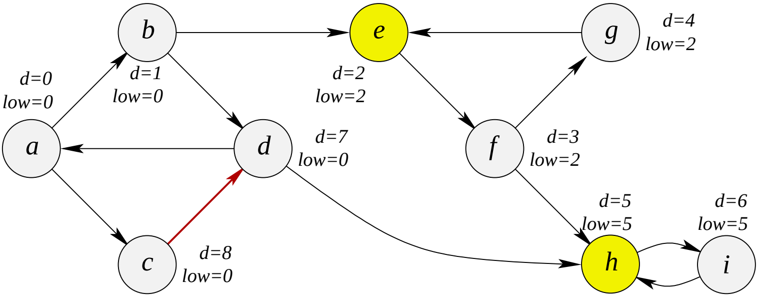
Algoritmo Tarjan



L: a, b, d, c

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

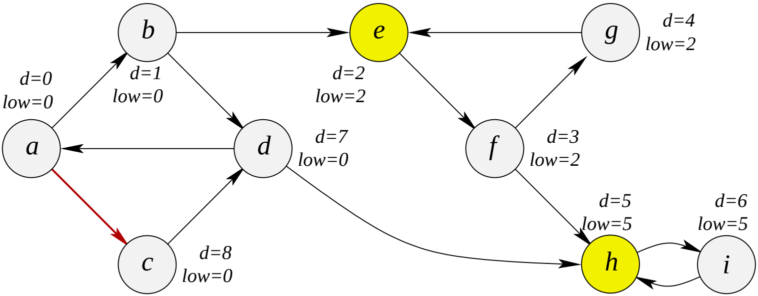
Algoritmo Tarjan



L: a, b, d, c

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

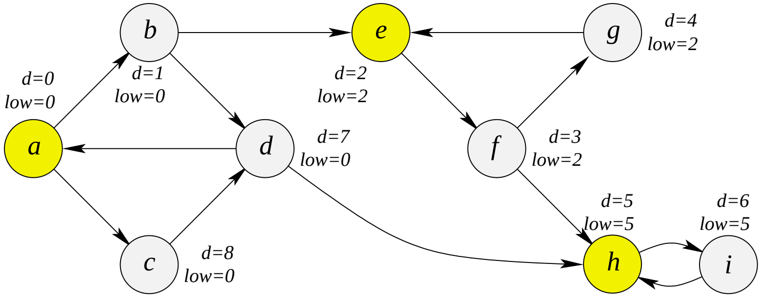
Algoritmo Tarjan



L: a, b, d, c

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

Algoritmo Tarjan



L:

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

Problema 1

Grafo Bipartido

Indique um algoritmo eficiente para determinar se um grafo $G = (V, E)$ é bipartido.

- Grafo G é bipartido se V pode ser dividido em L e R, tal que todos os arcos de G incidentes em 1 vértice de L e 1 vértice de R

Problema 2

Diâmetro de Árvore

Indique um algoritmo eficiente para calcular o diâmetro de uma árvore $T = (V, E)$

- Diâmetro: $max\{\delta(u, v) : u, v \in V\}$

Problema 2

Diâmetro de Árvore

Indique um algoritmo eficiente para calcular o diâmetro de uma árvore $T = (V, E)$

- Diâmetro: $max\{\delta(u,v) : u,v \in V\}$
- Solução: 2 BFS

Problema 3

Grafo Semi-Ligado

Indique um algoritmo eficiente para determinar se um grafo $G = (V, E)$ é semi-ligado.

- Um grafo dirigido $G = (V, E)$ diz-se semi-ligado se para qualquer par de vértices (u, v), u é atingível a partir de v ou v é atingível a partir de u