Strongly Connected Components with DFS

- Tarjan 1972 (wikipedia)
- Performs a single DFS with an additional Stack
Strongly connected components (SCC)

**Def:** Equivalence classes of the relation \((u, v)\) iff there is a path from \(u\) to \(v\) and a path from \(v\) to \(u\)

**Goal:** Compute for each vertex the number of its component
Strongly connected components (SCC)

Obs: Let $u$ and $v$ be in a SC $C$. Let $P$ be a path from $u$ to $v$ then every vertex on $P$ is also in $C$
DFS(G)
1. For each vertex \( u \in G. V \)
2. \( u.\text{color} = \text{WHITE} \)
3. \( u.\pi = \text{NIL} \)
4. \( \text{time} = 0 \)
5. For each vertex \( u \in G. V \)
6. if \( u.\text{color} == \text{WHITE} \)
7. \( \text{DFS-Visit}(G,u) \)

DFS-Visit(G,u)
1. \( \text{time} = \text{time} + 1 \)
2. \( u.d = \text{time} \)
3. \( u.\text{color} = \text{GRAY} \)
4. For each vertex \( v \in G. \text{Adj}[u] \)
5. if \( v.\text{color} == \text{WHITE} \)
6. \( v.\pi = u \)
7. \( \text{DFS-Visit}(G,v) \)
8. \( u.\text{color} = \text{BLACK} \)
9. \( \text{time} = \text{time} + 1 \)
10. \( u.f = \text{time} \)

**Thm 2:** \([v.d, v.f] \subset [u.d, u.f]\) iff \( v \) is a descendant of \( u \) in the DFS-forest, \([v.d, v.f] \cap [u.d, u.f] = \emptyset\) iff \( u \) and \( v \) are unrelated in the DFS-forest
DFS(G)
1. For each vertex $u \in G.V$
2. $u.\text{color} = \text{WHITE}$
3. $u.\pi = \text{NIL}$
4. $\text{time} = 0$
5. For each vertex $u \in G.V$
6. if $u.\text{color} == \text{WHITE}$
7. DFS-Visit(G,u)

DFS-Visit(G,u)
1. $\text{time} = \text{time} + 1$
2. $u.d = \text{time}$
3. $u.\text{color} = \text{GRAY}$
4. For each vertex $v \in G.\text{Adj}[u]$
5. if $v.\text{color} == \text{WHITE}$
6. $v.\pi = u$
7. DFS-Visit(G,v)
8. $u.\text{color} = \text{BLACK}$
9. $\text{time} = \text{time} + 1$
10. $u.f = \text{time}$

Thm 3: $v$ is a descendant of $u$ in the DFS-forest iff when we invoke DFS-Visit(u) there is a white path from $u$ to $v$
Edge classification

• Tree edges: \((u,v)\), DFS-Visit(v) was called from DFS-visit(u)
• Back edges: \((u,v)\) such that v is an ancestor of u in the DFS-forest
• Forward edges: nontree edges \((u,v)\) such that v is a descendant of u
• Cross edges: \((u,v)\) such that \(v.f < u.d\)

Observe, among non-tree edges:
1. (Only) backward edges go to a vertex with a later finish time
2. (Only) forward edges go to a vertex with later discovery time
Edge classification

- **Tree edges**: \((u,v)\), DFS-Visit\(v\) was called from DFS-visit\(u\)
- **Back edges**: \((u,v)\) such that \(v\) is an ancestor of \(u\) in the DFS-forest
- **Forward edges**: nontree edges \((u,v)\) such that \(v\) is a descendant of \(u\)
- **Cross edges**: \((u,v)\) such that \(v.f < u.d\)

Observe, among non-tree edges:

1. (Only) backward edges go to a vertex with a later finish time
2. (Only) forward edges go to a vertex with later discovery time
Tarjan’s algorithm

• We will find SCCs using only one DFS

Thm 9: If $u$ and $v$ are in the same SC then they are in same tree of the DFS-forest and $LCA(u, v)$ is also in this component

Corr: Each SC form a **connected** subtree of the DFS-forest
Proof: “by picture”, order the children of every node from left to right by their discovery time
Identify the roots of these subtrees
Use an additional stack ($S$)
Use an additional stack
Use an additional stack
Use an additional stack
Use an additional stack
Use an additional stack
Use an additional stack
Use an additional stack
Use an additional stack
Use an additional stack
Use an additional stack
Use an additional stack
Use an additional stack
Use an additional stack
Use an additional stack
Use an additional stack
Use an additional stack
Use an additional stack
Use an additional stack
Use an additional stack
Use an additional stack
The Stack Invariant

• Let $C_1, C_2, \ldots, C_i$ be the SC that intersect the path of the GRAY DFS vertices ordered top down.

• Then $S$ contains the vertices of each $C_j$ that we have already visited.

• The vertices of $C_1$ are at the bottom of $S$ and $\forall j > 1$ the vertices of $C_j$ are consecutive and on top of the vertices of $C_{j-1}$.

• $\forall j$ the root of $C_j$ is the bottommost vertex among all the vertices of $C_j$.

• Proof by induction on the steps of the algorithm.
Characterize these roots?

Thm 10: $v$ is a root of a component iff when the DFS backtracks from $v$ no back-edges or cross-edges going out of its subtree (in the DFS-forest) to vertices still in the stack.

Proof: Assume we apply this rule then the following Invariants hold:

- The components we have already identified are correct
- If we backtracked from $v$ and $v$ is not a root then $v$ is in the same component as its lowest GRAY ancestor
How do we identify roots?

For each vertex $v$ compute $low(v) = \text{the smallest } w \cdot d \text{ such that there is a cross edge or back edge to } w \in S \text{ out of the subtree of } v$. If there is no such edge then $low(v) = v \cdot d$.

If $low(v) = v \cdot d$ then $v$ is the root of a component.
How to compute low values?

Initialize $low(v) = v.d$

Update when you see a back edge or cross edge out of $v$

Update when you backtrack to $v$ from a child $w$ of $v$
Compute SCs using low values
Compute SCs using low values
SCs using low values
Tarjan’s SCC algorithm

SCC(G)
1. For each vertex \( u \in G. V \)
2. \( u.color = \text{WHITE} \)
3. \( u.\pi = \text{NIL} \)
4. \( \text{time} = 0 \)
5. For each vertex \( u \in G. V \)
6. if \( u.color == \text{WHITE} \)
7. SCC-Visit(G,u)

SCC-Visit(G,u)
1. \( \text{time} = \text{time} + 1 \)
2. \( u.d = \text{low}(u) = \text{time} \)
3. \( \text{push}(u,S) \)
4. \( u.color = \text{GRAY} \)
5. For each vertex \( v \in G. Adj[u] \)
6. if \( v.color == \text{WHITE} \)
7. \( v.\pi = u \)
8. SCC-Visit(G,v)
9. \( \text{low}(u) = \min\{\text{low}(u), \text{low}(v)\} \)
10. else if \( v \in S \)
11. \( \text{low}(u) = \min\{v.d, \text{low}(u)\} \)
12. \( u.color = \text{BLACK} \)
13. \( \text{time} = \text{time} + 1 \)
14. \( u.f = \text{time} \)
15. if \( \text{low}(u) = u.d \) then perform pop(S) until \( u \) is popped, the set popped is a SC