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

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

**Proof**: ...

**Corr**: Each SC form a **connected** subtree of the DFS-forest
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
אנחנו כמובן לא נסירים עם \( C_3 \) כל עוד הצומת הזה לבן.