

**STRONG COMPONENTS
IN A DIGRAPH**

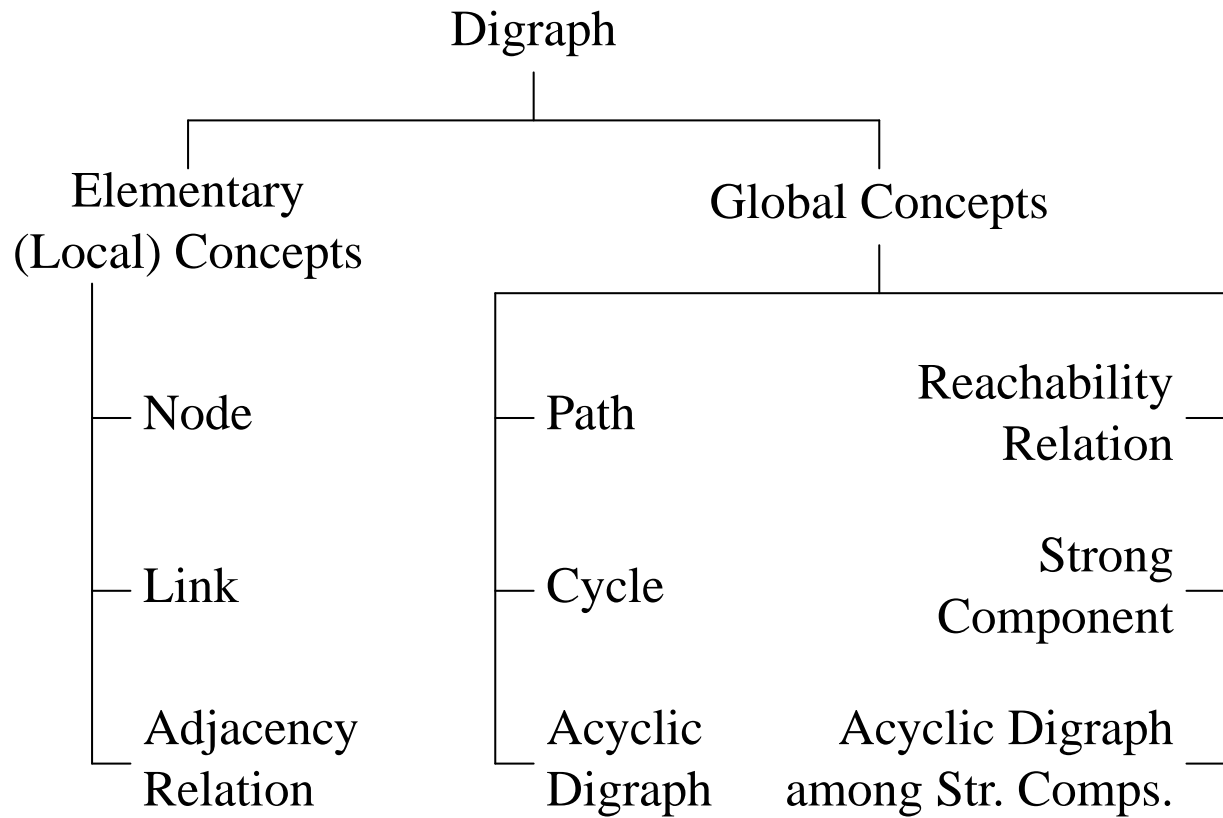
AND

THE DEPTH-FIRST ALGORITHM

FOR

**THEIR EFFICIENT
COMPUTATION**

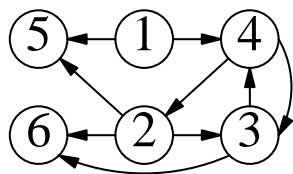
DIGRAPH CONCEPTS



PATHS AND CYCLES IN A DIGRAPH

Digraph $\vec{G} = (V, \vec{E})$:

- V is a set of *nodes* (vertices), and
- \vec{E} is a set of *directed edges* (x, y) ($\neq (y, x)$), also called *links* or *arcs*. The link (x, y) is adjacent *from* x to y .
- $N^+(x) = \{y: (x, y) \in \vec{E}\}$ and $\text{out-degree}(x) = |N^+(x)|$; x is *sink-node* if $N^+(x) = \emptyset$. Similarly, $N^-(x) = \{y: (y, x) \in \vec{E}\}$ and $\text{in-degree}(x) = |N^-(x)|$; x is a *source-node* if $N^-(x) = \emptyset$.
 $\sum |N^+(x)| = \sum |N^-(x)| = |\vec{E}|$.



- Source-node = 1; sink-nodes = 5 and 6.
- $\sum |N^+(x)| = 2+3+2+2+0+0 = 9 = |\vec{E}|$.
- Two cycles: $\langle 3, 4, 3 \rangle$ and $\langle 2, 3, 4, 2 \rangle$.

Complete Digraph \vec{K}_n : $V = \{1, 2, \dots, n\}$, $\vec{E} = \{(i, j): 1 \leq i \neq j \leq n\}$.

Path $\pi(x, y) = \langle x, x_1, x_2, \dots, x_n, y \rangle$ **from** x **to** y (xy -path):

- Each $(x_i, x_{i+1}) \in \vec{E}$, where $x_0 = x$ and $x_{n+1} = y$; x = start-node and y = end-node. Usually, all x_i will be distinct.

Cycle: A path starting and ending at the same node.

Acyclic \vec{G} : No cycles in \vec{G} (at most one xy -path for any $x \neq y$).

Questions: What is $\#(ij\text{-paths in } \vec{K}_n)$? What is $\#(\text{cycles in } \vec{K}_n)$?

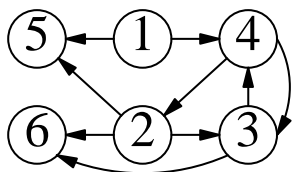
REACHABILITY RELATION AND STRONG CONNECTEDNESS

Reachability Relation:

- A node y is reachable from node x if there is an xy -path.
- $R(x) = \{y: y \text{ reachable from } x\}; x \in R(x)$.

Transitivity of Reachability Relation:

- Let $R(x, y) = y$ is reachable from x ; $R(x, x)$ for all x .
- If $R(x, y)$ and $R(y, z)$, then $R(x, z)$.



R	1	2	3	4	5	6
1	1	1	1	1	1	1
2	0	1	1	1	1	1
3	0	1	1	1	1	1
4	0	1	1	1	1	1
5	0	0	0	0	1	0
6	0	0	0	0	0	1

Strong Connectedness:

- There is an xy -path and an yx -path for each $x \neq y$. (This means every edge is in a cycle; the converse is not true!)
- $R(x) = V$ for each node x .

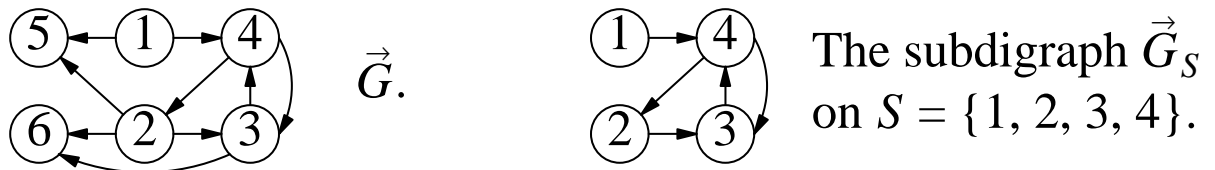


- Not strongly connected.

STRONG COMPONENTS OF A DIGRAPH

Subdigraph \vec{G}_S of \vec{G} on vertices $S \subseteq V$:

- $\vec{E}_S = \{(x, y) \in \vec{E} : \text{both } x \text{ and } y \in S\}$.
- If $S = \emptyset$, then \vec{G}_S is the empty digraph (no nodes and links).



Strong Component of \vec{G} : An equivalence class of $R \cap R^{-1}$.

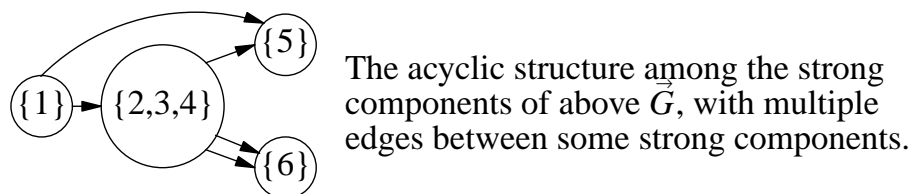
- A maximal strongly connected subdigraph \vec{G}_S .

Example. The above digraph has 4 strong components:

$$S_1 = \{1\}, S_2 = \{2, 3, 4\}, S_3 = \{5\}, S_4 = \{6\}.$$

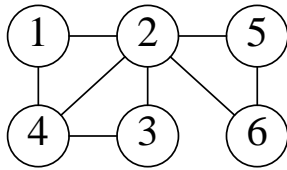
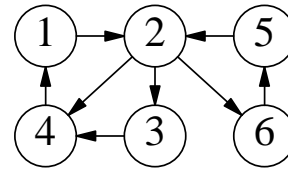
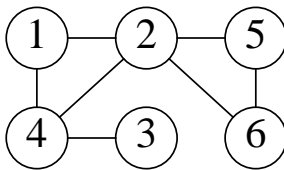
Comparison with Bicomponents in a Graph:

- Two strong components have no node in common. A link belongs to a cycle if and only if it is in a strong component.
- The strong components S_j in a digraph \vec{G} are connected with each other in the form of an *acyclic* digraph.



Question: How do the structure of components in a graph differ from that of strong components in a digraph?

CONVERTING A CONNECTED GRAPH INTO A STRONGLY CONNECTED DIGRAPH

(i) A graph G .(ii) A strongly connected orientation of G ; reversing all directions gives another one.

(iii) A graph that cannot be oriented to a strongly connected digraph.

Application: Orient each city-street in one-way without affecting the reachability among the nodes.

Question:

- ? How many ways can we orient the edges of G in (i) to form a strongly connected digraph?
- ? What property distinguishes the graphs in (i) and (iii) above that prevents a strongly connected orientation of the latter?

Theorem. If G is a connected graph with k bicomponents B_i (each B_i has ≥ 3 nodes) and $scoor(G) = \#(\text{strongly connected orientations of } G)$, then $scoor(G) = \prod_{i=1}^k scoor(B_i) \geq 2^k$.

Example. The graph G in (i) above has $k = 2$ bicomponents $B_1 = \{1, 2, 3, 4\}$ and $B_2 = \{2, 5, 6\}$, with $scoor(B_1) = 6$ and $scoor(B_2) = 2$. This gives $scoor(G) = 12 \geq 2^2$.

STRONGLY CONNECTED ORIENTATIONS OF A BICONNECTED GRAPH WITH ≥ 3 NODES

Some Equivalent Definitions of Strong Connectedness:

- (1) \vec{G} is strongly connected if and only if for some node x , there is an xy -path and an yx -path for each $y \neq x$.

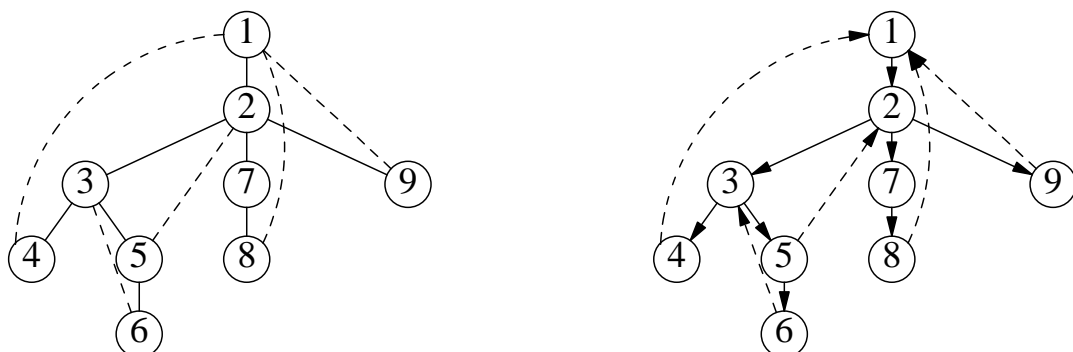
We can form an yz -path between any two nodes $y \neq z$ from the combination of an yx -path and an xz -path.

- (2) \vec{G} is strongly connected if and only if the following are true:
- (i) For some node x , there is an xy -path to each $y \neq x$.
 - (ii) Every edge of \vec{G} belongs to a cycle.

The conditions (i)-(ii) and $|V| > 1$ imply that every node of \vec{G} belongs to a cycle.

Key Idea Using (2):

- Choose a df-tree of G and direct the df-tree edges downward, giving a directed path from the root to each node.
- Direct the back-edges upwards to create directed cycles; each tree-edge and back-edge is now in a directed cycle.



Creating a strongly connected orientation of a biconnected graph with ≥ 3 nodes using a df-tree

DEPTH-FIRST TRAVERSAL OF A DIGRAPH

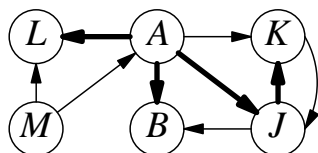
Same Basic Operations as Before:

(1) Move down a df-tree edge, (2) Backtrack, and (3) Terminate.

Three Key Differences:

- Each edge (x, y) is visited *at most once* from x to y , and each visited edge fall into one of four categories:
 - tree-edge: $y \in \text{children}(x)$.
 - forward-edge: $y \in \text{descendants}(x)$ and $y \notin \text{children}(x)$.
 - back-edge: $y \in \text{ancestors}(x)$; y may be $\text{parent}(x)$.
 - cross-edge: $y \notin \text{ancestors}(x)$ and $\text{dfLabel}(y) < \text{dfLabel}(x)$.
- $\#(\text{backtrackings to } x) = \#(\text{children of } x) \leq \text{out-degree}(x)$.

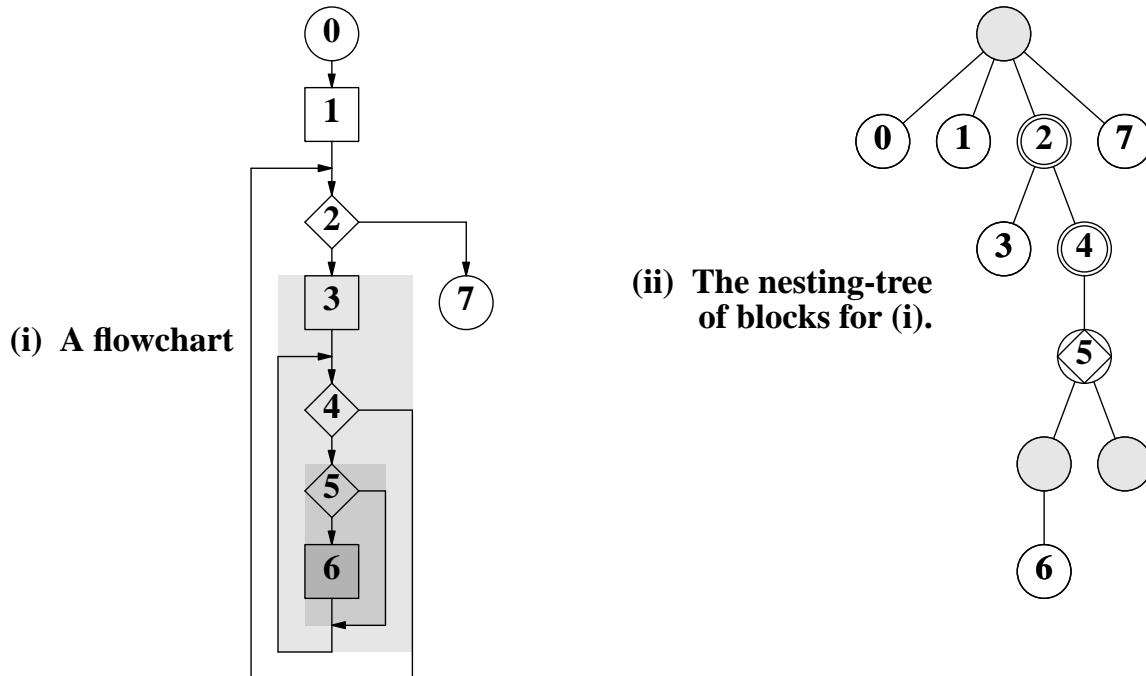
Example. Node M and the two edges from M are not visited when start-node $s = A$.



- $\text{adj}(A) = (B, J, K, L)$,
- $\text{adj}(B) = \emptyset$, etc;
- start-node $s = A$.

Current node y	Successive edges visited at various y				
	at A	at B	at J	at K	at L
$A, \text{dfLabel}(A) = 1$	(A, B) tree-edge				
$B, \text{dfLabel}(B) = 2$					
backtrack $B \rightarrow A$	(A, J) tree-edge				
$J, \text{dfLabel}(J) = 3$					
			(J, B) cross-edge		
			(J, K) tree-edge		
$K, \text{dfLabel}(K) = 4$					
backtrack $K \rightarrow J$				(K, J) back-edge	
backtrack $J \rightarrow A$	(A, K) forward-edge				
	(A, L) tree-edge				
$L, \text{dfLabel}(L) = 5$					
backtrack $L \rightarrow A$					
backtrack from A					

NESTING-TREE: AN INTERMEDIATE STEP TO DRAW FLOWCHART

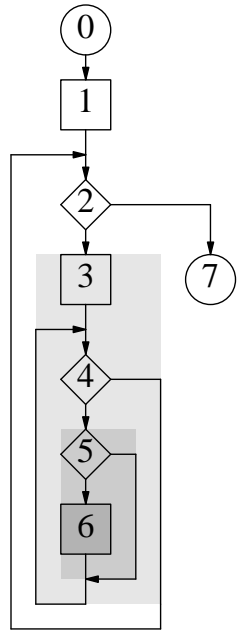


- There are 4 sequential "blocks" at the top-level, corresponding to the 4 children {0, 1, 2, 7} of root-node.
- The loop-body for while-do-test node 2 has two sequential blocks corresponding to the 2 children {3, 4} of node 2. (Similarly, for the loop-body for while-do-test node 4.)
- The two children of node 5 are place-holders for then-part and else-part of the if-test node 5.

Question:

- ? Show the df-tree for the second flowchart on page 1.12, with the true-branch at a branch-node followed first. What properties of a back-edge distinguish do-while vs. while-do loops?
- ? Show the nesting-tree for the second flowchart on page 1.12. How can you determine this from df-tree?

SEQUENTIAL-BLOCK IN A STRUCTURED FLOWCHART



- A structured flowchart, made of if-then if-then-else, while-do, and do-while. – no goto's or break-statements.
- A semi-structured flowchart may have break-statements.
- Some of the sequential blocks:
 $B(5) = \{5, 6\}$ and $B(2) = \{2, 3, 4, 5, 6\}$.

Sequential Block $B(x)$ Starting at a Node x :

- (1) $x \in B(x)$ and each node $y \in B(x)$ is reachable from x .
- (2) If (u, v) is a link from a node $u \notin B(x)$ to a node $v \in B(x)$, then $v = x$. That is, x is the unique entry node to $B(x)$.
- (3) If (y, z) and (y', z') are links from nodes $y, y' \in B(x)$ to nodes $z, z' \notin B(x)$, then $z = z'$. That is, all links leaving $B(x)$ do to the same (unique exit-to) node.
- (4) $B(x)$ is minimal with properties (1)-(3).

Question:

- ? Why is $\{5\}$ not a sequential-block? What is the node x for the sequential-block $\{4, 5, 6\}$? Given a sequential block B , how can we find x such that $B = B(x)$? Is x always unique?
- ? Does $y \in B(x)$ imply $B(y) \subset B(x)$? Justify your answer.

DEPTH-FIRST FOREST FOR A DIGRAPH

df-Tree: A rooted ordered tree, with root = start-node and the edges $\{(x, y): y \text{ is reached first time via edge } (x, y)\}$. Children of a node are ordered by their dfLabels.

df-Forest: Successively choose a start-node $s = x_i$ not yet visited and do a df-traversal.

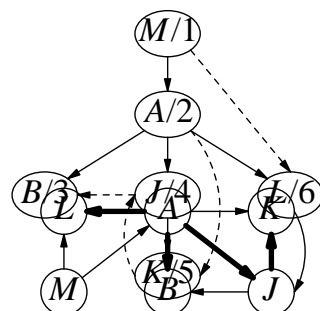
Back-edge (x, y) : $\text{dfLabel}(y) < \text{dfLabel}(x)$ and not backtracked from y yet.

Cross-edge (x, y) : $\text{dfLabel}(y) < \text{dfLabel}(x)$ and y already backtracked from.

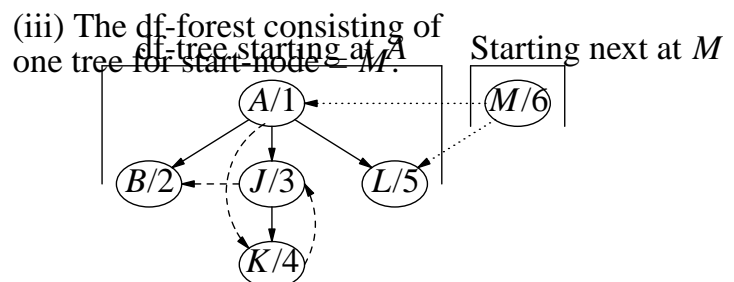
Forward-edge (x, y) : $\text{dfLabel}(y) > \text{dfLabel}(x)$ and $x \neq \text{parent}(y)$.

Example. A digraph and two df-forests for it.

- Df-forest in (ii) has two df-trees; first start-node $x_1 = A$ visits $\{A, B, J, K, L\}$ and next start-node $x_2 = M$ visits $\{M\}$.
- The df-forest in (iii) has one df-tree, with $x_1 = M$.



(i) A digraph \vec{G} .



(ii) Back-edge (K, J) , cross-edges $\{(J, B), (M, A), (M, L)\}$, and forward-edge (A, K)

DF-SEARCH IN A DIGRAPH FOR NODES REACHABLE FROM A START-NODE

Notes:

- Backtracking corresponds to return from the recursion.
- The recursive-call at x for a start-node s creates the subtree at x in the resulting df-tree(s).

Algorithm DF-SEARCH(currNode): //fi rst call: currNode = s

Input: The adjacency-lists $\text{adj}(x)$ and the start-node s .

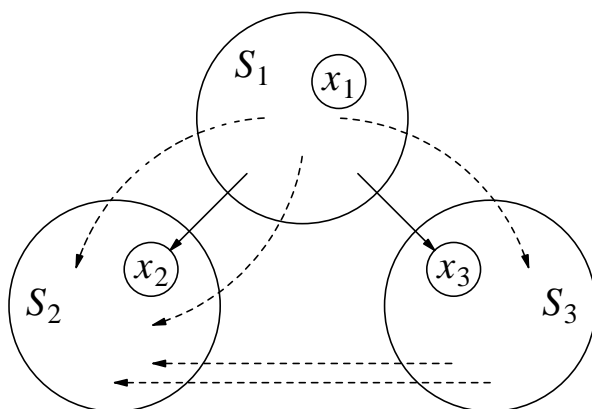
Output: The df-tree, with root = s .

1. [Create df-tree node.] Create currTreeNode in df-tree for currNode. If ($\text{lastDfLabel} = 0$), then let $\text{dfLabel}(x) = 0$ for each x , $\text{parent}(\text{currTreeNode}) = \text{currTreeNode}$, and $\text{lastDfLabel} = 0$. Now, let $\text{dfLabel}(\text{currNode}) = \text{lastDfLabel} = \text{lastDfLabel} + 1$.
2. For (each node x in $\text{adj}(\text{currNode})$) do the following:
 - If ($\text{dfLabel}(x) = 0$) then add the root-node (which corresponds to x) of the subtree returned by $\text{DF-SEARCH}(x)$ as the next child of currTreeNode.
3. Return(currTreeNode).

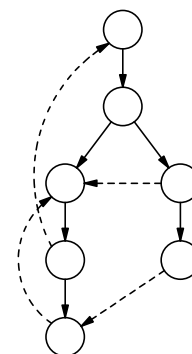
Complexity: $O(|V| + |\vec{E}|)$ for all searches combined, where we initiate a new fresh call by choosing a new start-node $s = x_i$ which is not yet visited. Each edge is visited exactly once.

BACK-EDGES vs. STRONG COMPONENTS

- Each back-edge (x, y) forms a (directed) cycle with the yx -path in df-tree and hence is in some strong component \vec{S}_i . Also, each cycle must contain at least one back-edge.
- The forward-edges do not play any role; their deletion does not affect the strong components.
- A strong-component \vec{S}_i other than the one containing the start-node s is entered just once via a tree-edge ($\text{parent}(x_i), x_i$) and all of \vec{S}_i is visited before backtracking from x_i .
 - No back-edge from (descendants of) x_i to ancestors of x_i .
- There can be cross- and forward-edges from nodes in \vec{S}_i to nodes within and outside of \vec{S}_i .
- A cross-edge (x, y) can be part of a cycle \vec{C} only if \vec{C} contains one or more back-edges.
- Once backtracked from x_i and detected \vec{S}_i , cross edges into \vec{S}_i from nodes outside \vec{S}_i are not in a strong component.



Cross-edges
and forward-
edges
between an
 \vec{S}_i and \vec{S}_j .



Cross-edges
and back-
edges
within an \vec{S}_i
forming
cycles

A NEW NOTION OF $\text{LOW}(y)$ FOR DETECTING STRONG COMPONENTS

- For a start-node s , consider only the new nodes visited since s , excluding nodes reached from previous start-nodes, if any.
- S = the stack of nodes x visited for a given start-node and which are not yet assigned to a strong component (x added to S when x is assigned dfLabel and deleted from S when we output strong component containing x).

Low(y):

$$\begin{aligned}
 &= \min \left\{ \begin{array}{l} \text{df-label}(z): \quad z = y \text{ or} \\ \text{there is a back/cross-edge from } y \\ \text{or from a descendant of } y \text{ to } z \in S \end{array} \right\} \\
 &= \min \left\{ \begin{array}{l} \text{df-label}(y), \\ \min \{ \text{df-label}(z): (y, z) \text{ is a back/cross-edge to } z \in S \}, \\ \min \{ \text{low}(y_i): y_i \text{ is a child of } y \text{ in df-tree} \} \end{array} \right\}
 \end{aligned}$$

- $\text{low}(y)$ is known when we backtrack from y .

Detection of Entry-Nodes x_i for a Strong Component \vec{S}_i :

- $\text{low}(x_i) = \text{dfLabel}(x_i)$ when we backtrack from x_i .

Detection of Strong Component \vec{S}_i :

- Output and delete all nodes from S upto and including x_i .

MODIFICATION TO DF-SEARCH FOR COMPUTING $\text{low}(x)$ AND STRONG COMPONENTS

- $\text{dfLabel}(x) = -1$ means x is removed from stack S .
- $\text{backTracked}(x) = 1$ means x has been backtracked from.

Computing $\text{low}(x)$:

1. [Initialize $\text{low}(x)$.] As you assign $\text{dfLabel}(x) \geq 1$, let $\text{low}(x) = \text{dfLabel}(x)$ and add x to stack S .
2. [Update $\text{low}(x)$ for a back-edge.] When a back-edge (x, y) is detected, i.e., $0 < \text{dfLabel}(y) < \text{dfLabel}(x)$ and $\text{backTracked}(y) = 0$, let $\text{low}(x) = \min \{ \text{low}(x), \text{dfLabel}(y) \}$.
3. [Update $\text{low}(x)$ for a cross-edge within a strong-component.] When such a cross-edge (x, y) is detected, i.e., $0 < \text{dfLabel}(y) < \text{dfLabel}(x)$, $\text{backTracked}(y) = 1$, and $y \in S$, let $\text{low}(x) = \min \{ \text{low}(x), \text{dfLabel}(y) \}$.
3. [Update $\text{low}(x)$ as you backtrack to x from a child y of x by setting $\text{backTracked}(y) = 1$.] Let $\text{low}(x) = \min \{ \text{low}(x), \text{low}(y) \}$. (No need to test if $y \in S$; if $y \notin S$, then $\text{low}(y) = \text{dfLabel}(y) > \text{dfLabel}(x)$.)

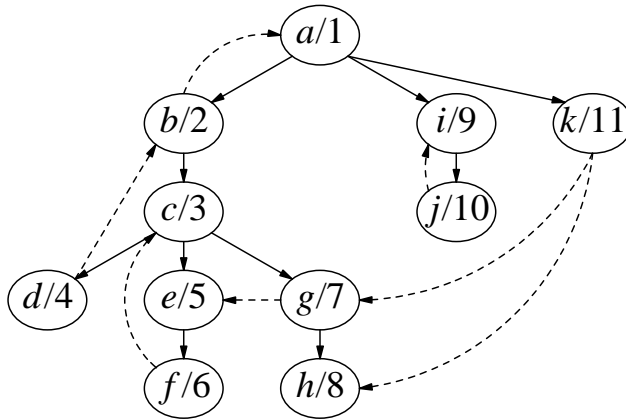
Outputting Strong Component:

- If $\text{low}(x) = \text{dfLabel}(x)$ as we backtrack from x , then output all items y in S upto x (including x) and set $\text{dfLabel}(y) = -1$.

Choosing Next Start-node:

- This is the first node x with $\text{backTracked}(x) = 0$; keep a static variable `firstNodeNotBacktracked` to avoid searching from the beginning of backtracked-array every time.

EXAMPLE OF STRONG COMPONENT COMPUTATION



$$\begin{aligned} \text{low}(d) &= \min \{4, 2\} = 2. \\ \text{low}(f) &= \min \{6, 3\} = 3. \\ \text{low}(e) &= \min \{5, 3\} = 3. \\ \text{low}(g) &= \min \{7, 5\} = 5. \end{aligned}$$

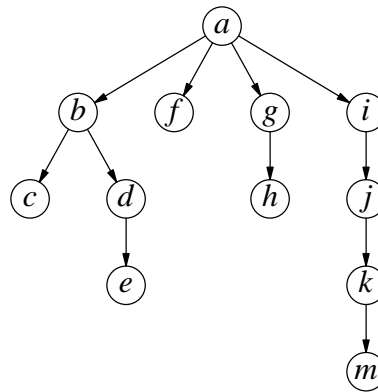
$$\begin{aligned} \text{low}(c) &= \min \{3, \text{low}(d), \text{low}(e), \text{low}(g)\} \\ &= \{3, 2, 3, 5\} = 2. \end{aligned}$$

$$\text{low}(k) = \min \{11, 7\} = 7.$$

Link processed	DfLabel	Low	Stack and str-component
start at a	$\text{dfLabel}(a) = 1$	$\text{low}(a) = 1$	a
(a, b)	$\text{dfLabel}(b) = 2$	$\text{low}(b) = 2$	b, a
(b, a)		$\text{low}(b) = 1$	
(b, c)	$\text{dfLabel}(c) = 3$	$\text{low}(c) = 3$	c, b, a
(c, d)	$\text{dfLabel}(d) = 4$	$\text{low}(d) = 4$	d, c, b, a
(d, b)		$\text{low}(d) = 2$	
backtrack $d \rightarrow c$	$\text{low}(d) \neq \text{dfLabel}(d)$	$\text{low}(c) = 2$	
(c, e)	$\text{dfLabel}(e) = 5$	$\text{low}(e) = 5$	e, d, c, b, a
(e, f)	$\text{dfLabel}(f) = 6$	$\text{low}(f) = 6$	f, e, d, c, b, a
(f, c)		$\text{low}(f) = 3$	
backtrack $f \rightarrow e$	$\text{low}(f) \neq \text{dfLabel}(f)$	$\text{low}(e) = 3$	
backtrack $e \rightarrow c$	$\text{low}(e) \neq \text{dfLabel}(e)$	$\text{low}(c) = 2$	
(c, g)	$\text{dfLabel}(g) = 7$	$\text{low}(g) = 7$	g, f, e, d, c, b, a
(g, e)	cross-edge within $\text{strComp}(g)$	$\text{low}(g) = 5$	
(g, h)	$\text{dfLabel}(h) = 8$	$\text{low}(h) = 8$	h, g, f, e, d, c, b, a
backtrack $h \rightarrow g$	$\text{low}(h) = \text{dfLabel}(h)$	$\text{low}(g) = 5$	$g, f, e, d, c, b, a;$ {h}
backtrack $g \rightarrow c$	$\text{low}(g) \neq \text{dfLabel}(g)$	$\text{low}(c) = 2$	
backtrack $c \rightarrow b$	$\text{low}(c) \neq \text{dfLabel}(c)$	$\text{low}(b) = 1$	
backtrack $b \rightarrow a$	$\text{low}(b) \neq \text{dfLabel}(b)$	$\text{low}(a) = 1$	
(a, i)	$\text{dfLabel}(i) = 9$	$\text{low}(i) = 9$	i, g, f, e, d, c, b, a
(i, j)	$\text{dfLabel}(j) = 10$	$\text{low}(j) = 10$	$j, i, g, f, e, d, c, b, a$
(j, i)		$\text{low}(j) = 9$	
backtrack $j \rightarrow i$	$\text{low}(j) \neq \text{dfLabel}(j)$	$\text{low}(i) = 9$	
backtrack $i \rightarrow a$	$\text{low}(i) = \text{dfLabel}(i)$	$\text{low}(a) = 1$	$g, f, e, d, c, b, a;$ {j, i}
(a, k)	$\text{dfLabel}(k) = 11$	$\text{low}(k) = 11$	k, g, f, e, d, c, b, a
(k, g)		$\text{low}(k) = 7$	
(k, h)	cross-edge outside $\text{strComp}(k)$		
backtrack $k \rightarrow a$	$\text{low}(k) \neq \text{dfLabel}(k)$	$\text{low}(a) = 1$	
backtrack $a \rightarrow \text{out}$	$\text{low}(a) = \text{dfLabel}(a)$		$\emptyset;$ {k, g, f, e, d, c, b, a}

Questions:

- ? Show the maximum number of back-edges, forward-edges, and cross-edges in a digraph \vec{G} with the following df-tree and the strong components $\{b, c, d, e\}$, $\{a, f, g, h\}$, $\{i, j\}$, and $\{k, m\}$. Also show the minimum number of the same.



- ? Let C be a strong component in \vec{G} . How do you determine the vertices s such that a df-traversal from s will enter C ?
- ? Give a sufficient condition on \vec{G} so that for each start-vertex s each strong component will be entered (if at all) through the same vertex in each df-traversal?
- ? Suppose some df-traversal of \vec{G} enters a strong component C through the vertex x . What properties of strong components imply that on backtracking from x we would have visited all vertices of C ? What properties of x tell us that it is the vertex through which C was entered?
- ? Why is it necessary to ignore the vertices of strong components that have been already outputted in the computation of low-values? Explain with an example what would go wrong if we did not ignore those vertices.

DETECTING EDGES IN A STROG COMOPNENT THAT CAN BE DELETED KEEPING IT STRONGLY CONNECTED

- We need to keep at most one back-edge (x, y) from a node x , namely, y with smallest $dfLabel$.