Graph search algorithms

Combinatorial optimization

Giovanni Righini

Università degli Studi di Milano

Breadth-first search

Given:

- a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$
- a vertex $s \in \mathcal{V}$,

we indicate with $\mathcal{V}_k$ the set of all vertices that

- are reachable from s along a path made of k edges;
- and not reachable from s along any path with less than k edges.

Recursive definition:

- $\mathcal{V}_0 = \{s\}$
- $\mathcal{V}_{k+1} = \{v \in \mathcal{V} \setminus \bigcup_{i=0}^{k} \mathcal{V}_i : \exists u \in \mathcal{V}_k \wedge \exists [u, v] \in \mathcal{E}\}$.

Analogous definitions hold for digraphs.

Breadth-first search

To compute $\mathcal{V}_{k+1}$ it is enough to scan the set of all edges (arcs) incident to (leaving) the vertices (nodes) in $\mathcal{V}_k$ e to insert these vertices (nodes) into $\mathcal{V}_{k+1}$, if they have not been reached before. A binary flag associated with each vertex (node) is enough to check this.

The complexity of this algorithm is $O(m)$, because each edge (arc) is scanned at most twice (once).

This BFS algorithm determines the shortest path from s to any other vertex (node) of the (di-)graph in the special case of unit weight edges (arcs).

Pseudo-code

Breadth-First Search (Berge 1958, Moore 1959):

begin
for v:=1 **to** n **do** flag[v]:=0; flag[s]:=1;
$k := 0$; $\mathcal{V}_k := \{s\}$;
while $\mathcal{V}_k \neq \emptyset$ **do**
    $\mathcal{V}_{k+1} := \emptyset$;
    for $u \in \mathcal{V}_k$ **do**
        for $[u, v] \in \delta(u)$ **do**
        if (flag[v]=0) **then**
            $\mathcal{V}_{k+1} := \mathcal{V}_{k+1} \cup \{v\}$;
            flag[v] := 1;
    $k := k + 1$;
end.

The vertices (nodes) not reached when the algorithm terminates do not belong to the same connected component of s.

Connected components

Corollary (Shirey, 1969). The connected components of $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ can be computed in linear time.

Depth-First Search (Tarry 1895)

Given:

- a digraph $\mathcal{D} = (\mathcal{N}, \mathcal{A})$
- a node $s \in \mathcal{N}$,

we define *Scan*(s) the following recursive procedure:

for $(s, v) \in \delta^+(s)$ **do**
    for $(u, v) \in \delta^-(v) : u \neq s$ **do**
        Delete (u, v);
    Scan(**v**);

If all nodes in $\mathcal{N}$ are reachable from s, the arcs not deleted by *Scan*(s) form an arborescence rooted in s and spanning them.

Depth-First Search

To implement the *Delete* operation we can associate a binary flag "existing (1)/deleted (0)" with each arc.

Pseudo-code of $DFS(root)$:

```
for (i,j) ∈ A do
    Flag[(i,j)] ← 1
Scan(root)
```

Pseudo-code of $Scan(i)$:

```
for (i,j) ∈ δ+(i) do
    if Flag(i,j) = 1 then
        for (k,j) ∈ δ−(j) do
            if k ≠ i then
                Flag[(i,j)] ← 0
        Scan(j)
```

Depth-First Search

A slightly different implementation of DFS requires a binary flag for each node, meaning “visited (1)/not visited (0)”.

Pseudo-code of *DFS*(*root*):

```
for i = 1,...,n do
    Flag[i] ← 0
Scan(root)
```

Pseudo-code of *Scan*(*i*):

```
Flag[i] ← 1
for (i,j) ∈ δ⁺(i) do
    if Flag[j] = 0 then
        Scan(j)
```

Complexity

If the graph is represented as an adjacency matrix, then DFS takes $O(n^2)$, because all cells of the matrix need to be tested or modified, including those that do not correspond to existing arcs.

If the graph is represented with out-stars and in-stars, then its complexity can be reduced to $O(m)$.

To achieve this with the first version, it is necessary that

- either a single record is used to represent each arc and it is linked in bi-dimensional linked list (rows = in-stars; columns = out-stars)
- or there is a pair of pointers between the two records corresponding to the same arc (in the in-star of the head and in the out-star of the tail).

Each arc is considered at most twice, as a member of an in-star and of an out-star and the operations take $O(1)$ for each arc.

(Pre-)topological order

The nodes of a digraph are sorted in topological order if $i < j \ \ \forall (v_i, v_j) \in \mathcal{A}$.

Hence a subset $\mathcal{N}'$ of nodes can be sorted in topological order only if the induced subgraph $(\mathcal{N}', \mathcal{A}(\mathcal{N}'))$ is acyclic (i.e. it does not contain circuits).

The nodes of a digraph are sorted in pre-topological order if the following condition holds:

$$v_i \prec v_j \Rightarrow i < j$$

dove $v_i \prec v_j$ means that j is reachable from i but i is not reachable from j.

If the digraph is acyclic, then any pre-topological order is also topological.

Pre-topological order

Theorem. Given a di-graph $\mathcal{D} = (\mathcal{N}, \mathcal{A})$ and a node $s \in \mathcal{N}$, the nodes in $\mathcal{N}$ reachable from s can be sorted in pre-topological order in $O(m')$, where m' is the number of arcs reachable from s.

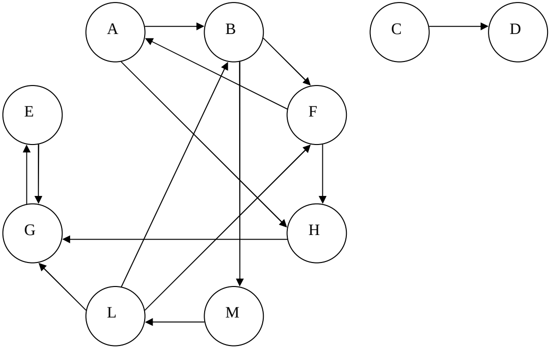
Proof. In the execution of **Scan(s)** all nodes reachable from s are scanned. The order in which their **Scan()** procedure *terminates* is the reverse of their pre-topological order. For each pair of nodes u and v reachable from s, if there is a path from u to v but not from v to u, then **Scan(v)** terminates before **Scan(u)**.

Corollary 1. The nodes of a digraph $\mathcal{D}(\mathcal{N}, \mathcal{A})$ can be sorted in pre-topological order in linear time.

Proof. Insert a dummy node s into the digraph together with arcs $(s, v)\ \forall v \in \mathcal{N}$ and then apply the previous theorem.

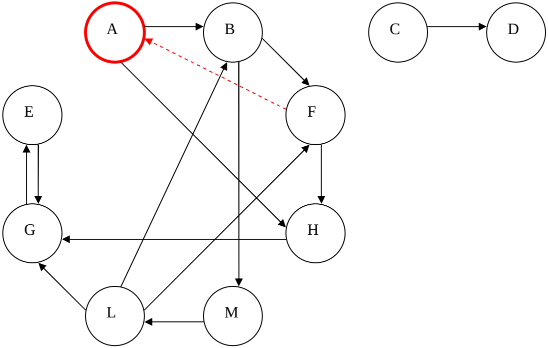
Corollary 2. The nodes of an acyclic digraph $\mathcal{D}(\mathcal{N}, \mathcal{A})$ can be sorted in topological order in linear time.

Example



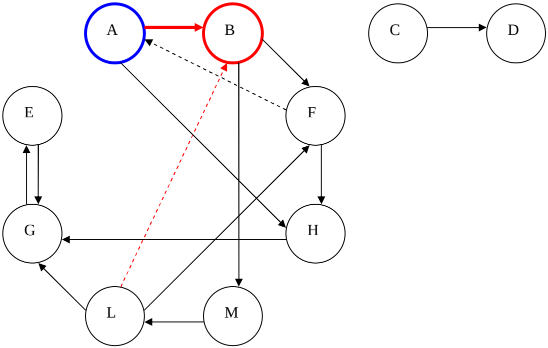
Scan										
End										
Order	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**

Example



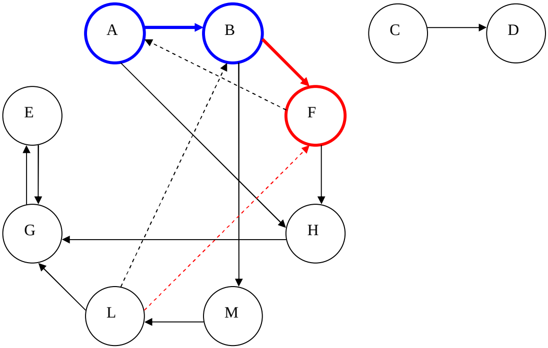
Scan	**A**									
End										
Order	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**

Example



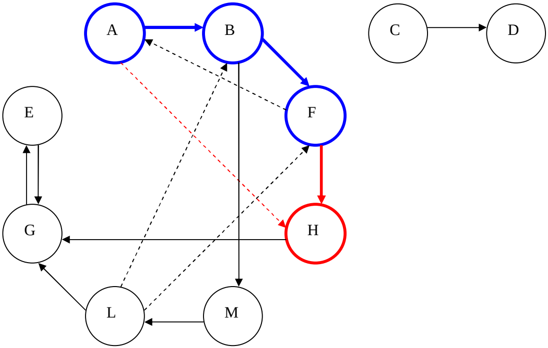
Scan	**A**	**B**								
End										
Order	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**

Example



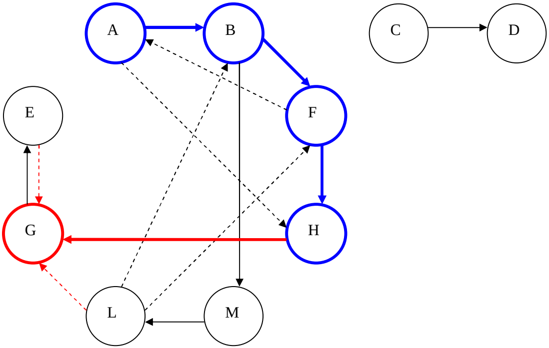
Scan	**A**	**B**	**F**							
End										
Order	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**

Example



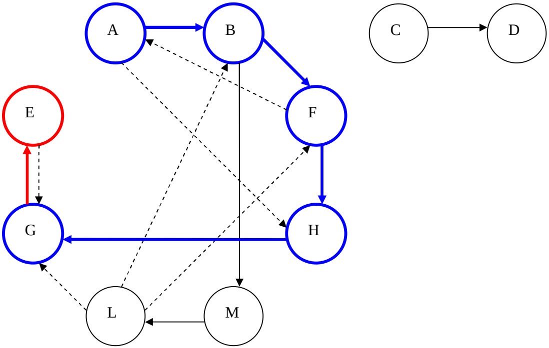
Scan	**A**	**B**	**F**	**H**						
End										
Order	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**

Example



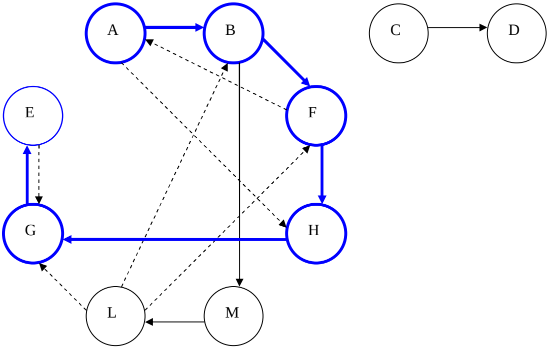
Scan	**A**	**B**	**F**	**H**	**G**					
End										
Order	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**

Example



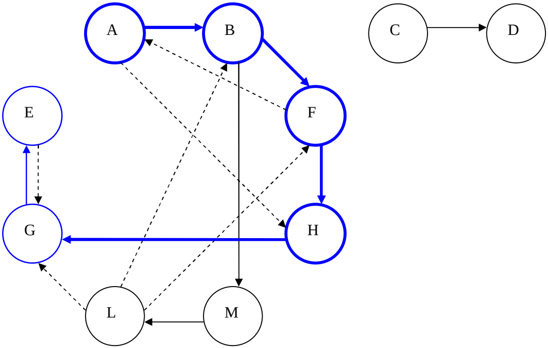
Scan	**A**	**B**	**F**	**H**	**G**	**E**				
End										
Order	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**

Example



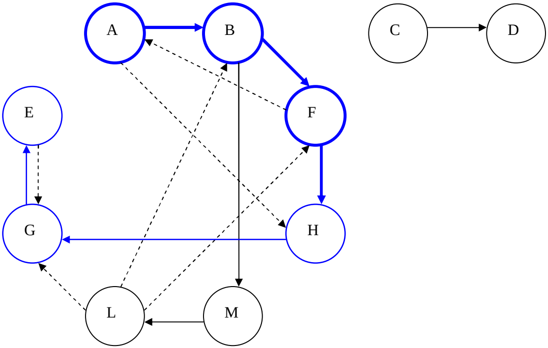
Scan	**A**	**B**	**F**	**H**	**G**	**E**				
End	**E**									
Order	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**

Example



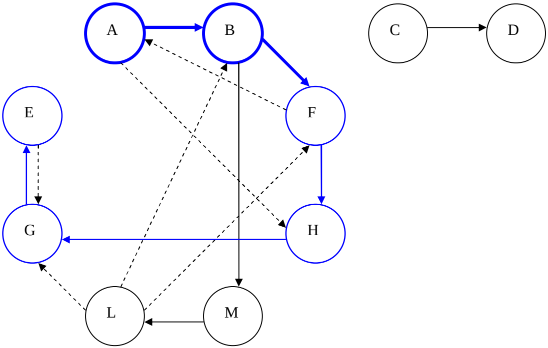
Scan	**A**	**B**	**F**	**H**	**G**	**E**				
End	**E**	**G**								
Order	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**

Example



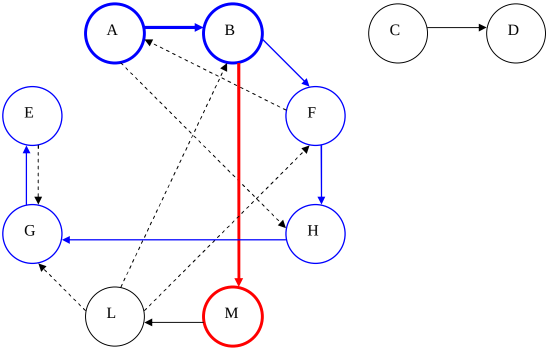
Scan	**A**	**B**	**F**	**H**	**G**	**E**				
End	**E**	**G**	**H**							
Order	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**

Example



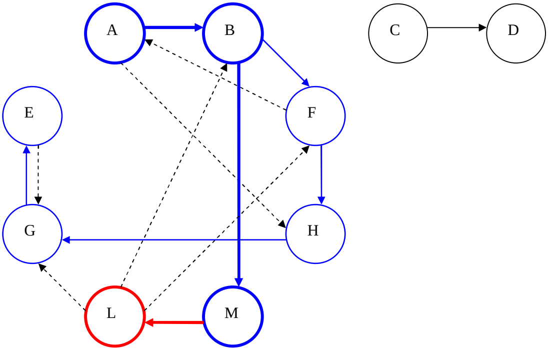
Scan	**A**	**B**	**F**	**H**	**G**	**E**				
End	**E**	**G**	**H**	**F**						
Order	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**

Example



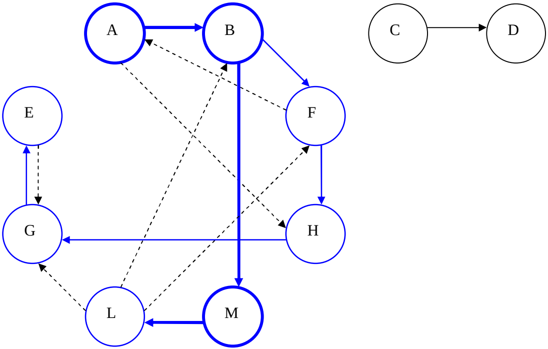
Scan	**A**	**B**	**F**	**H**	**G**	**E**	**M**			
End	**E**	**G**	**H**	**F**						
Order	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**

Example



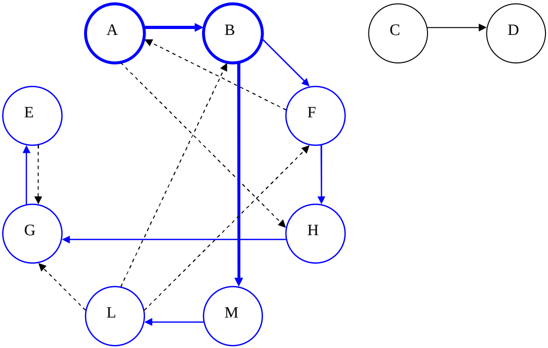
Scan	**A**	**B**	**F**	**H**	**G**	**E**	**M**	**L**		
End	**E**	**G**	**H**	**F**						
Order	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**

Example



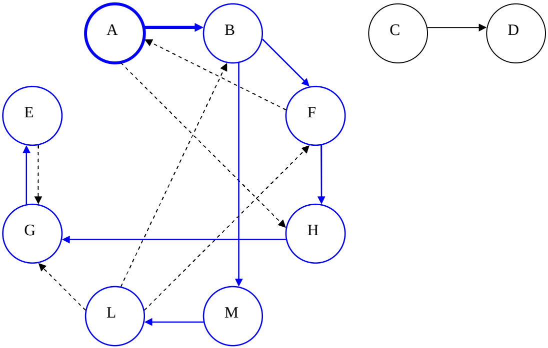
Scan	**A**	**B**	**F**	**H**	**G**	**E**	**M**	**L**		
End	**E**	**G**	**H**	**F**	**L**					
Order	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**

Example



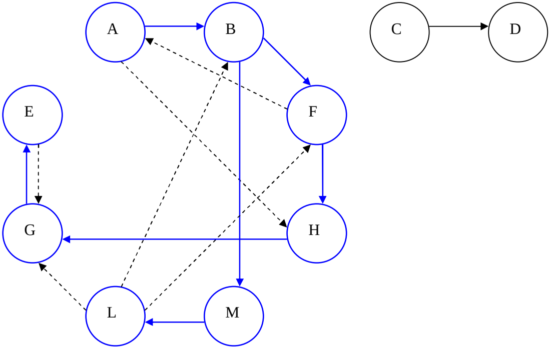
Scan	**A**	**B**	**F**	**H**	**G**	**E**	**M**	**L**	
End	**E**	**G**	**H**	**F**	**L**	**M**			
Order	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**

Example



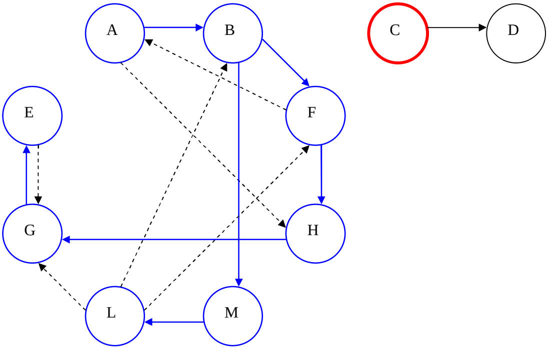
Scan A	B	F	H	G	E	M	L		
End E	G	H	F	L	M	B			
Order 10	9	8	7	6	5	4	3	2	1

Example



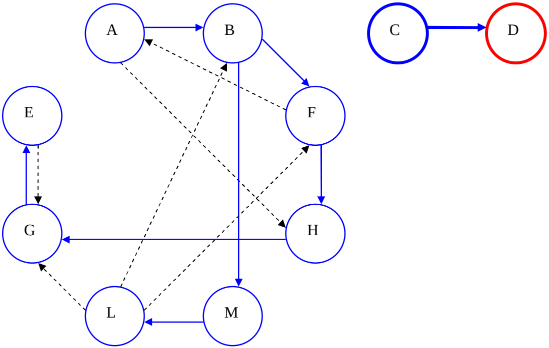
Scan	**A**	**B**	**F**	**H**	**G**	**E**	**M**	**L**		
End	**E**	**G**	**H**	**F**	**L**	**M**	**B**	**A**		
Order	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**

Example



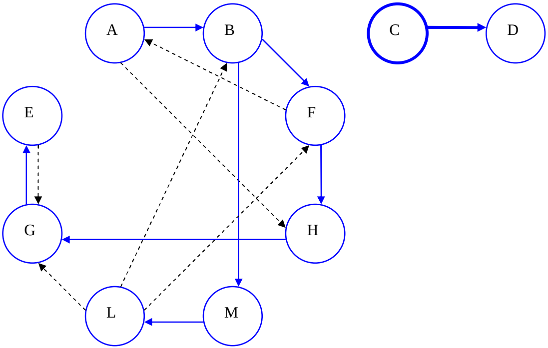
Scan	**A**	**B**	**F**	**H**	**G**	**E**	**M**	**L**	**C**	
End	**E**	**G**	**H**	**F**	**L**	**M**	**B**	**A**		
Order	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**

Example



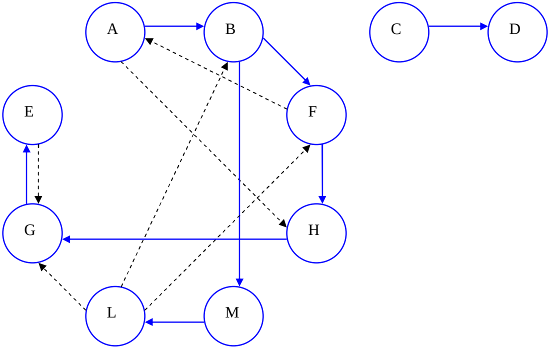
Scan	**A**	**B**	**F**	**H**	**G**	**E**	**M**	**L**	**C**	**D**
End	**E**	**G**	**H**	**F**	**L**	**M**	**B**	**A**		
Order	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**

Example



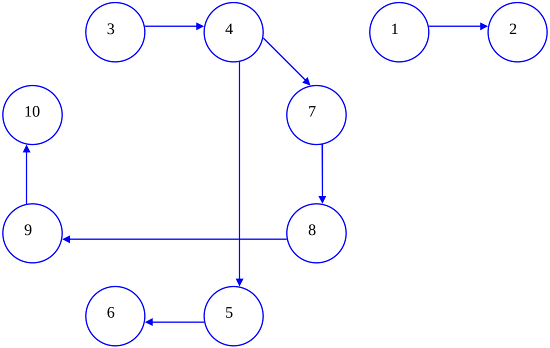
Scan	**A**	**B**	**F**	**H**	**G**	**E**	**M**	**L**	**C**	**D**
End	**E**	**G**	**H**	**F**	**L**	**M**	**B**	**A**	**D**	
Order	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**

Example



Scan	**A**	**B**	**F**	**H**	**G**	**E**	**M**	**L**	**C**	**D**
End	**E**	**G**	**H**	**F**	**L**	**M**	**B**	**A**	**D**	**C**
Order	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**

Example



Scan	**A**	**B**	**F**	**H**	**G**	**E**	**M**	**L**	**C**	**D**
End	**E**	**G**	**H**	**F**	**L**	**M**	**B**	**A**	**D**	**C**
Order	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**

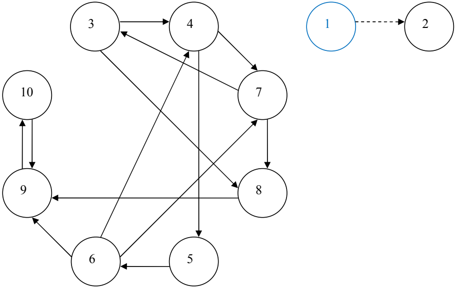
Strongly connected components

Theorem (Kosaraju e Sharir, 1981). Given a digraph $\mathcal{D} = (\mathcal{N}, \mathcal{A})$ its strongly connected components (s.c.c.) can be computed in linear time.

Proof. Sort the nodes in pre-topological order: $v_1, v_2, \ldots, v_n$. Let $\mathcal{N}_1$ be the set of nodes from which v_1 is reachable. Then $\mathcal{N}_1$ is the s.c.c. v_1 belongs to: each $v_j \in \mathcal{N}_1$ is reachable from v_1 for the pre-topological order properties.
For the previous theorem $\mathcal{N}_1$ can be computed in $O(|\mathcal{A}_1|)$ time (with DFS on the reversed arcs) where $\mathcal{A}_1$ is the set of arcs with their head in $\mathcal{N}_1$.
Deleting all nodes in $\mathcal{N}_1$ and the arcs in $\mathcal{A}_1$ another digraph is obtained whose nodes are sorted in pre-topological order in the same sequence as before.
Therefore, by repeatedly applying the procedure, all s.c.c. are obtained.

Example

In our example node 1 (originally node C) is the first in the pre-topological order. Running DFS from 1 with reversed arcs, we see that there are no predecessors.

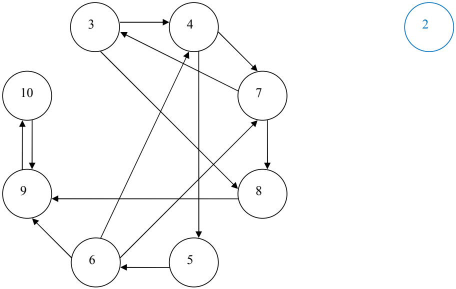


Scan	**1**
End	**1**

Hence $\mathcal{V}_1 = \{1\}$.

Example

Now we consider node 2 and the same happens.

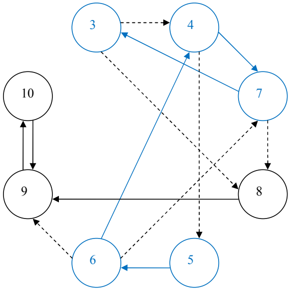


Scan	**2**
End	**2**

Hence $\mathcal{V}_2 = \{2\}$.

Example

Now we consider node 3 (originally node A). Running DFS from 3 with reversed arcs, we visit some nodes.

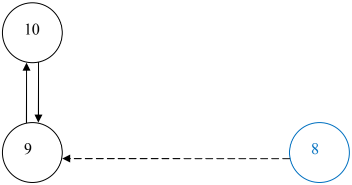


Scan	**3**	**7**	**4**	**6**	**5**
End	**5**	**6**	**4**	**7**	**3**

Hence $\mathcal{V}_3 = \{3, 4, 5, 6, 7\}$.

Example

Running DFS from node 8 with reversed arcs, we find no predecessors.

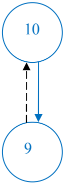


Scan	**8**
End	**8**

Hence $\mathcal{V}_4 = \{8\}$.

Example

Finally we consider node 9 and we run DFS from 9 with reversed arcs:



Scan	**9**	**10**
End	**10**	**9**

Hence $\mathcal{V}_5 = \{9, 10\}$ and the algorithm is over. Five s.c.c. have been detected.