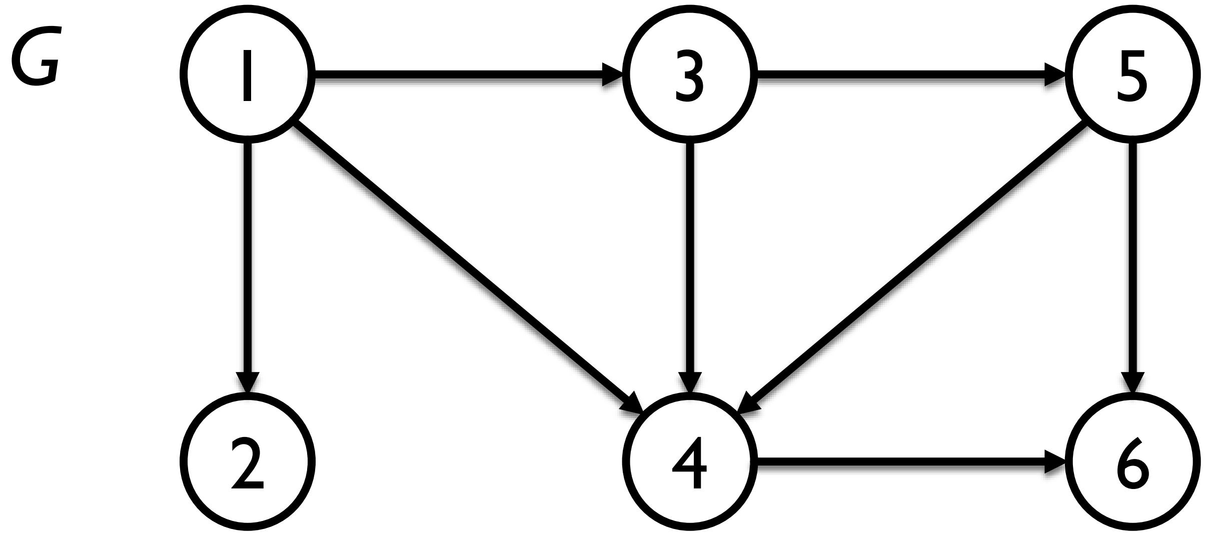
ADVANCED ANALYSIS OF ALGORITHMS
CPS 5440

GRAPH ALGORITHMS. TOPOLOGICAL SORT.

TOPOLOGICAL SORT

- **Sorting technique over DAGs (Directed Acyclic Graphs)**
- **It creates a linear sequence (ordering) for the nodes such that:**
  - **If u has an outgoing edge to v ➔ then u must finish before v starts**
- **Very common in ordering jobs or tasks**

TOPOLOGICAL SORT: EXAMPLE



$T = (1, 3, 2, 5, 4, 6)$

Note: there may be multiple topological orderings.

$T = (1, 2, 3, 5, 4, 6)$ is also valid.

TOPOLOGICAL SORT: PRACTICAL EXAMPLE

A practical example of topological sorting is a list of tasks that needs to be completed, with some tasks having to be completed first.
The tasks would be nodes in a graph.

<u>A practical example:</u>

A cooking recipe.

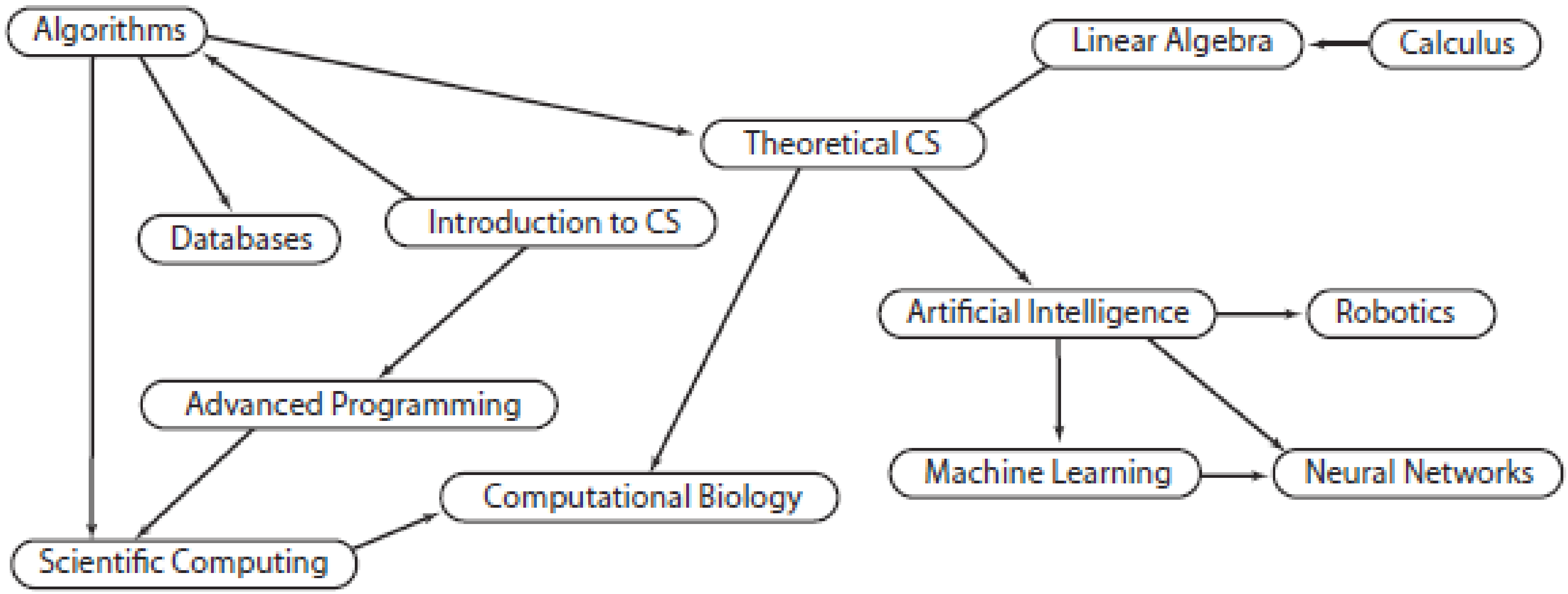
You need to crack eggs before you can beat them, you must preheat the oven before you put food in it, etc.

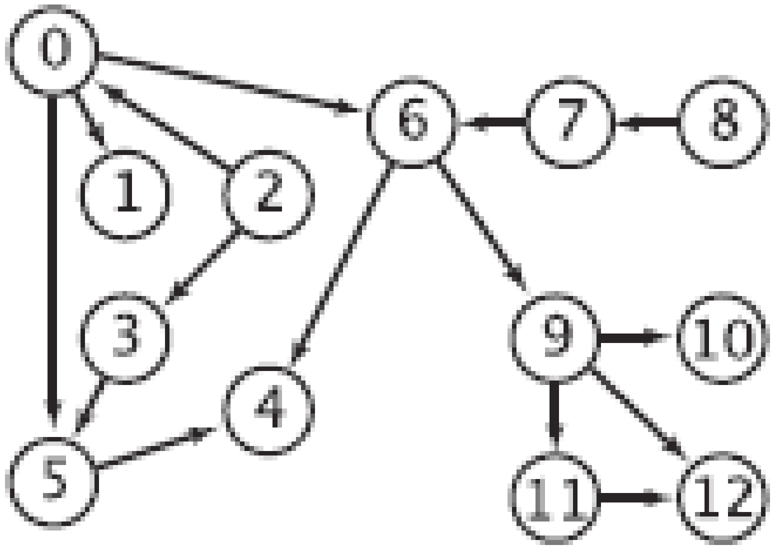
TOPOLOGICAL SORT: PRACTICAL EXAMPLE

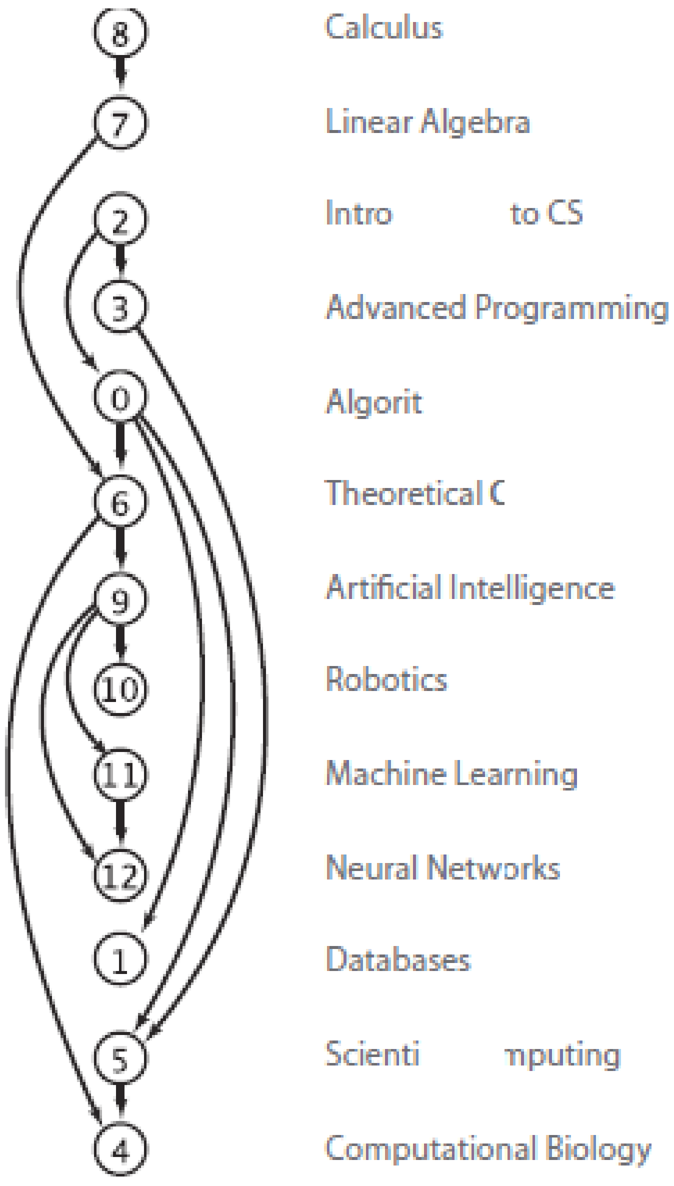
Given a list of lectures that John wants to attend and the prerequisite lectures for each lecture, construct a list of the order in which John should attend the lectures.

The lectures are vertices of a directed graph, and the prerequisites are directed edges of the graph. The topological sorting of the graph provides the list that is required.

TOPOLOGICAL SORT: PRACTICAL EXAMPLE





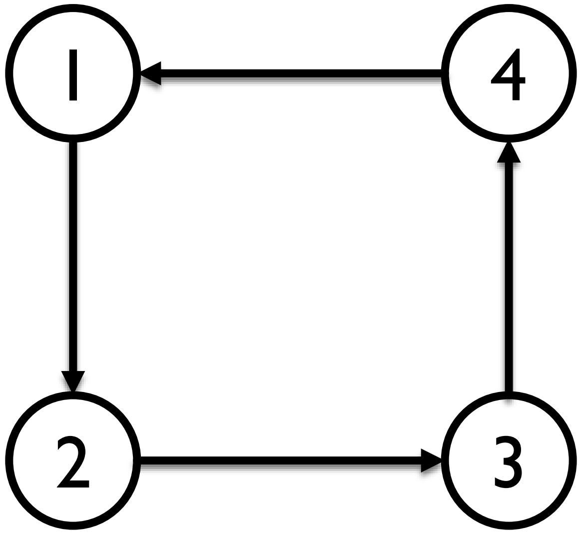


TOPOLOGICAL SORT: PRACTICAL EXAMPLE

- School class prerequisites
- Program dependencies
- Event scheduling
- Assembly instructions
- Etc …

TOPOLOGICAL SORT: CONDITION

A topological ordering of a graph is possible if and only if the graph does not contain any directed cycles.



(A topological ordering of this graph does not exist.)

There are two main algorithms for finding the topological order of a graph:

1. Kahn's Algorithm
2. DFS

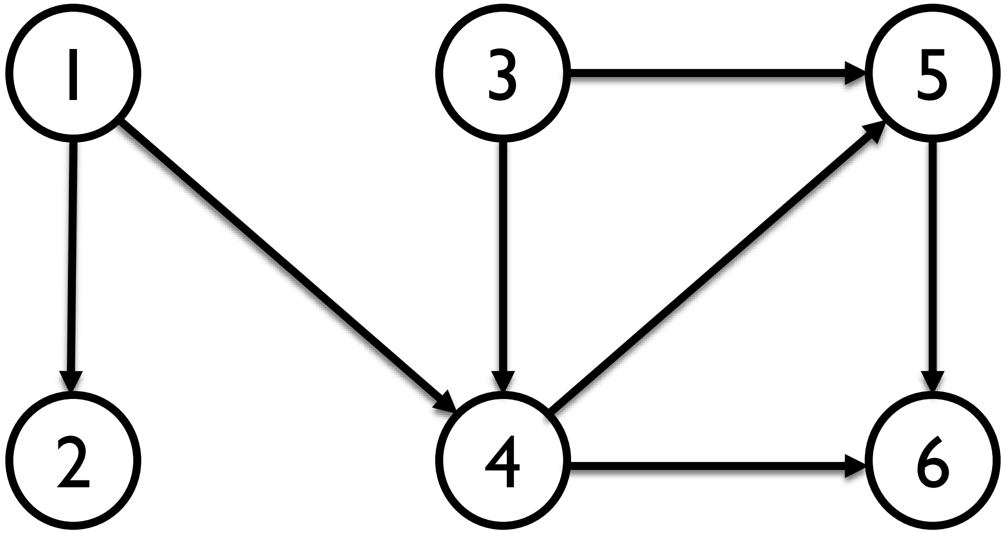
A DAG G has <u>at least</u> one vertex with in-degree 0 and one vertex with out-degree 0.

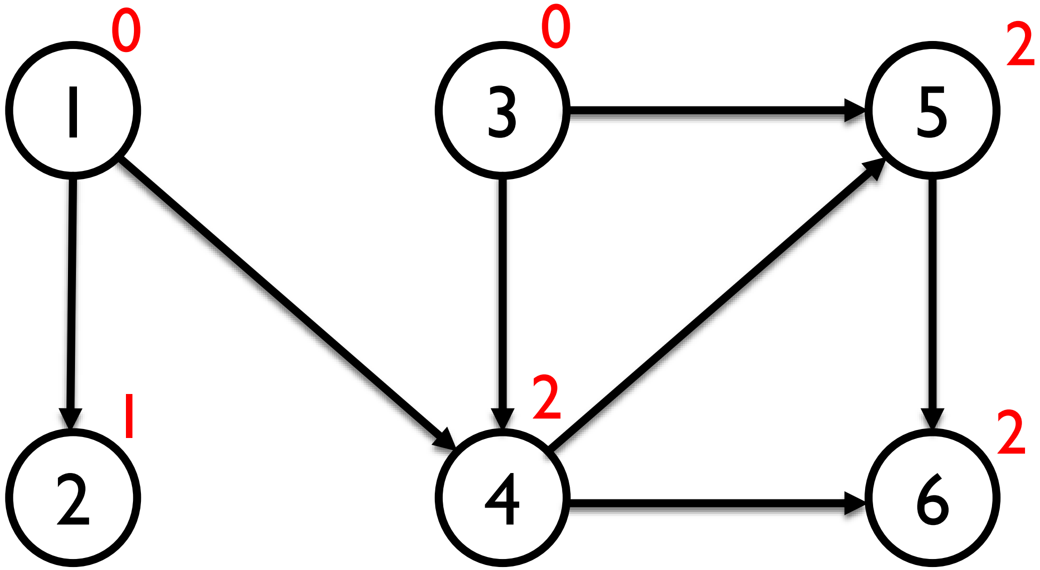
Proof: There's a simple proof to the above fact is that a DAG does not contain a cycle which means that all paths will be of finite length. Now let S be the longest path from u(source) to v(destination). Since S is the longest path there can be no incoming edge to u and no outgoing edge from v, if this situation had occurred then S would not have been the longest path
$\Rightarrow indegree(u) = 0$ and $outdegree(v) = 0$

TOPOLOGICAL SORT: KAHN'S ALGORITHM

1. Compute the in-degree of each vertex
2. Add all the vertices with an in-degree of 0 onto a queue Q
3. Remove a vertex V from Q
4. Increment the counter of visited nodes by 1
5. Add V onto T where T is the list that will contain the order of the nodes
6. Decrease the in-degree of all neighbours of V by 1
7. If a neighbour now has an in-degree of 0, add it to Q
8. If Q is not empty, go to step 3
9. If the vertex counter does not equal the total number of vertices, return $ERROR$ (the topological ordering does not exist)
10. Return T

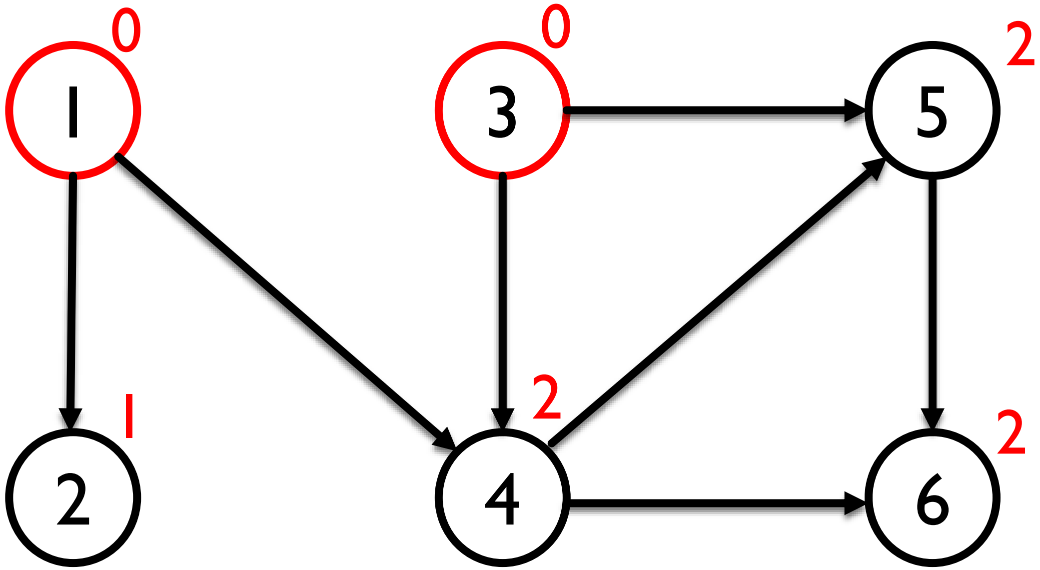
TOPOLOGICAL SORT: KAHN'S ALGORITHM – EXAMPLE





Compute the in-degree of each vertex.

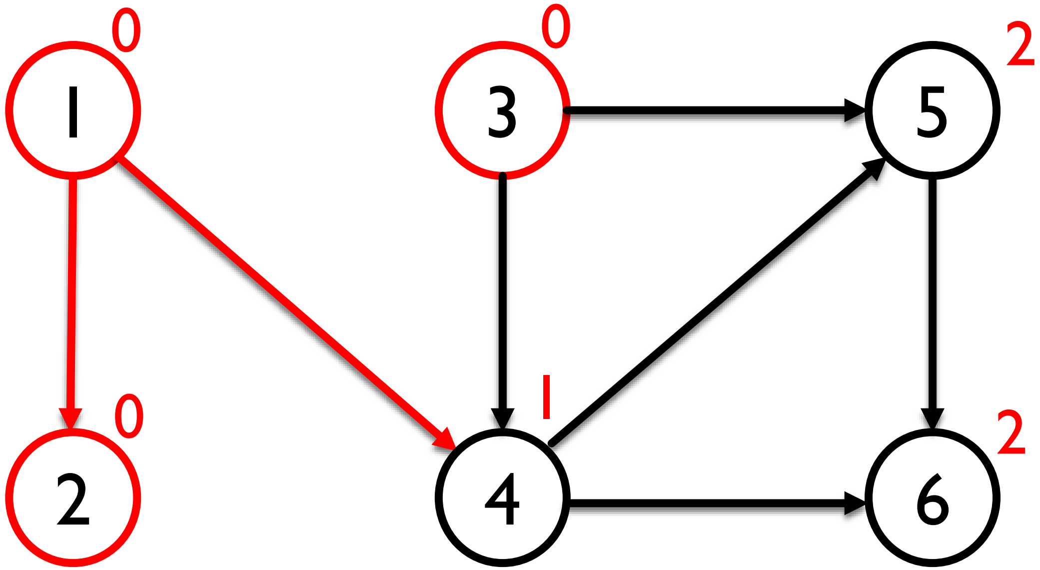
TOPOLOGICAL SORT: KAHN'S ALGORITHM – EXAMPLE



Put the vertices with in-degree 0 onto a queue.

$$Q = (1, 3)$$
$$T = ()$$

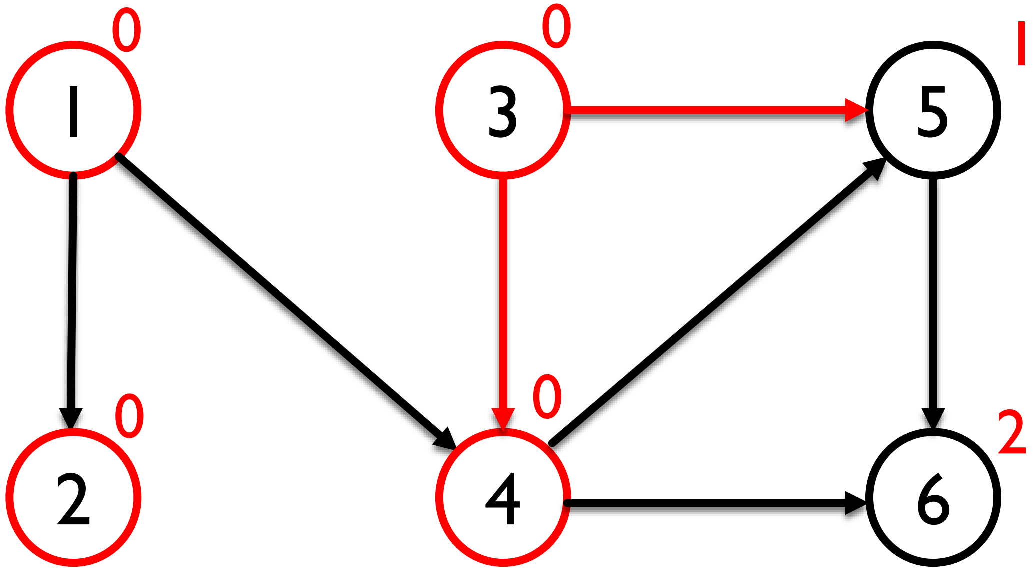
TOPOLOGICAL SORT: KAHN'S ALGORITHM – EXAMPLE



Add a vertex to T and reduce the in-degree of its neighbours by 1.

$$Q = (3, 2)$$
$$T = (1)$$

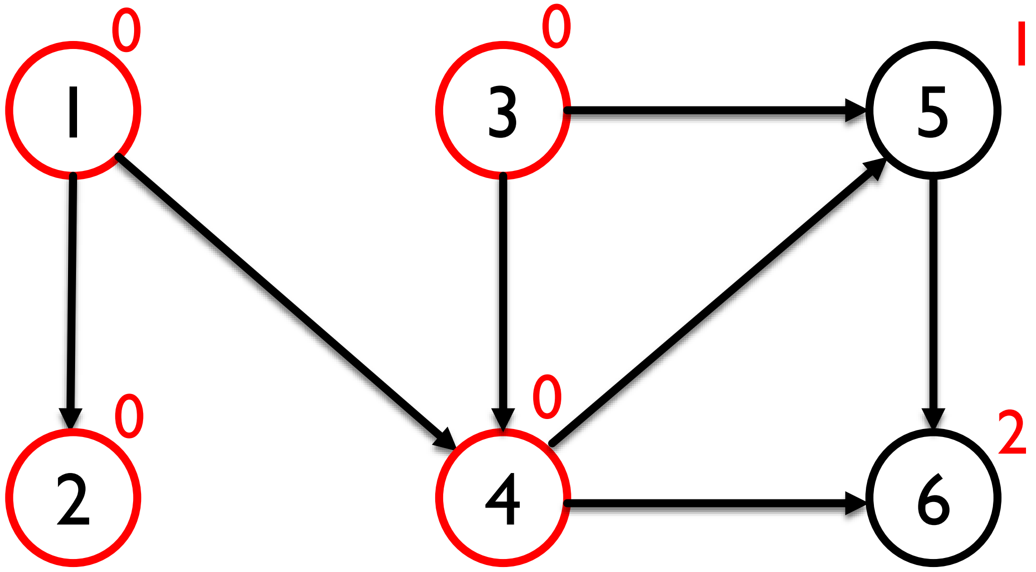
TOPOLOGICAL SORT: KAHN'S ALGORITHM – EXAMPLE



$$Q = (2, 4)$$
$$T = (1, 3)$$

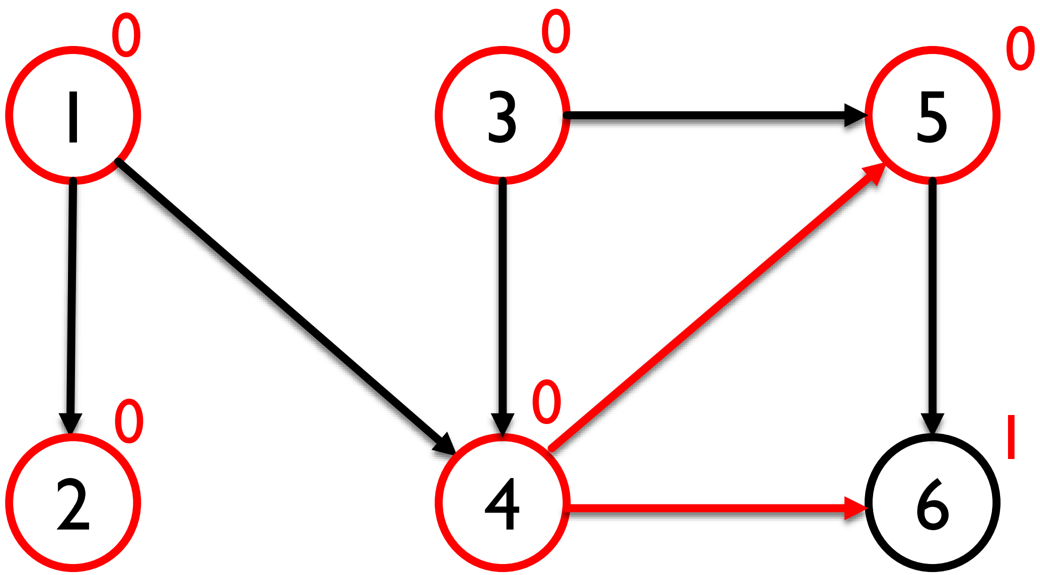
Add a vertex to T and reduce the in-degree of its neighbours by 1.

TOPOLOGICAL SORT: KAHN'S ALGORITHM – EXAMPLE



Add a vertex to T and reduce the in-degree of its neighbours by 1.

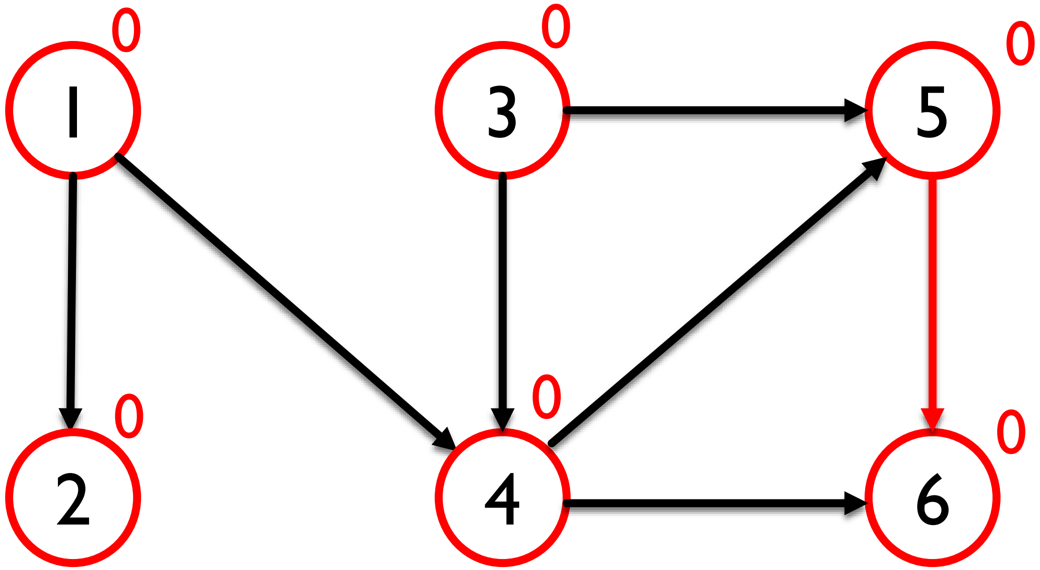
$$Q = (4)$$
$$T = (1, 3, 2)$$



Add a vertex to T and reduce the in-degree of its neighbours by 1.

$$Q = (5)$$
$$T = (1, 3, 2, 4)$$

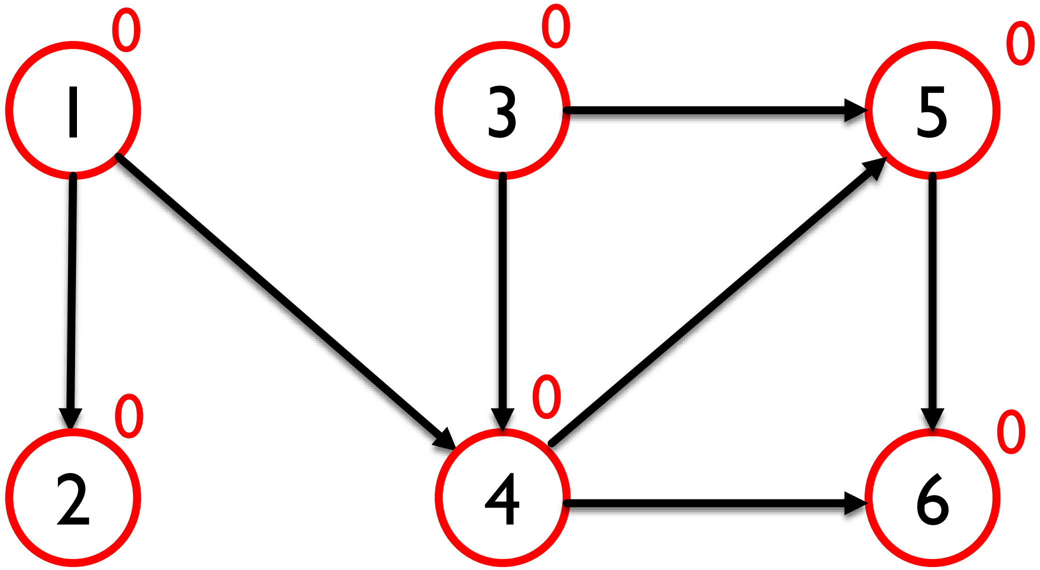
TOPOLOGICAL SORT: KAHN'S ALGORITHM – EXAMPLE



Add a vertex to T and reduce the in-degree of its neighbours by 1.

$$Q = (6)$$
$$T = (1, 3, 2, 4, 5)$$

TOPOLOGICAL SORT: KAHN'S ALGORITHM – EXAMPLE



Return T.

$$Q = ()$$
$$T = (1, 3, 2, 4, 5, 6)$$

TOPOLOGICAL SORT: KAHN'S ALGORITHM – COMPLEXITY

- Time-complexity: $O(V + E)$

You go through each vertex once, and you check each edge once.

- Space-complexity: $O(V + E)$

You only need a list containing the vertices and a list containing the edges.

TOPOLOGICAL SORT - DFS

- To create a topological sort from a DAG

  1- Final linked list is empty

  2- Run DFS

  3- When a node becomes black (finishes), insert it to the top of a linked list

TOPOLOGICAL SORT - DFS EXAMPLE

A job consists of 10 tasks with the following precedence rules:

Must start with 7, 5, 4 or 9.

Task 1 must follow 7.

Tasks 3 & 6 must follow both 7 & 5.

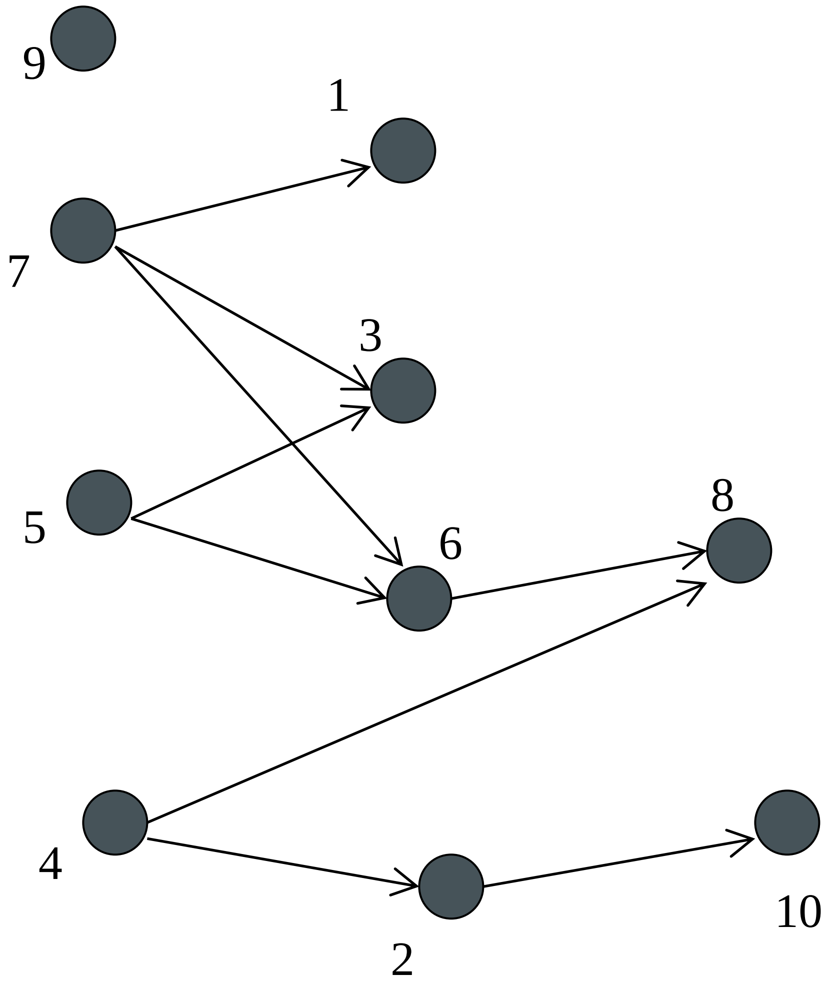
8 must follow 6 & 4.

2 must follow 4.

10 must follow 2.

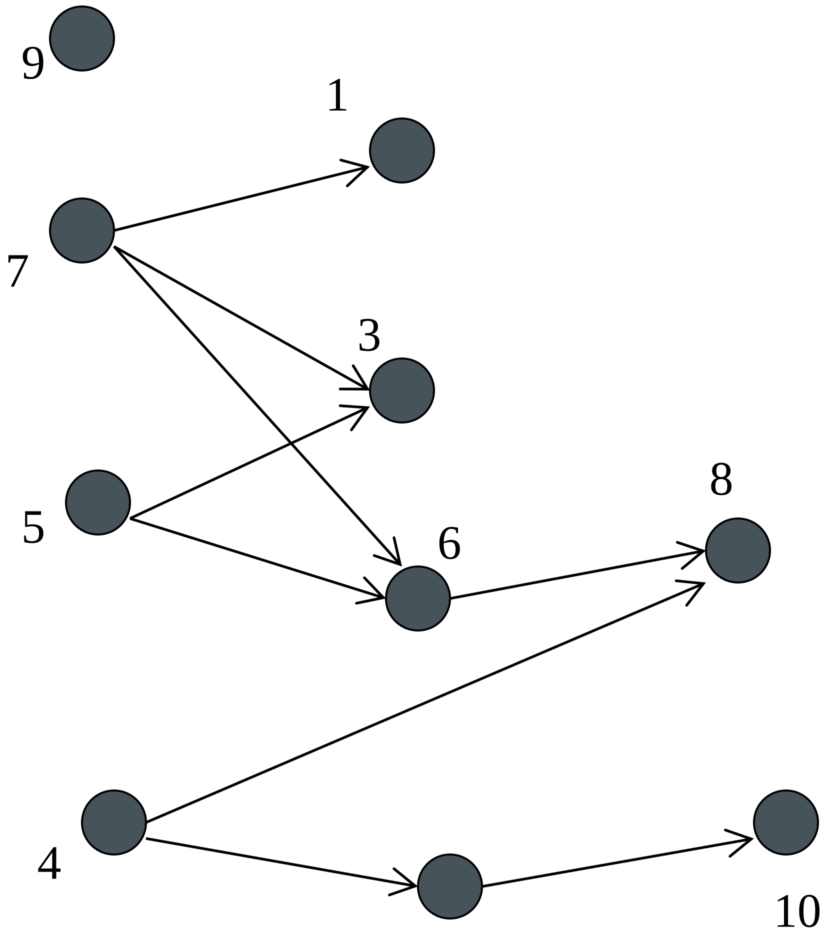
Make a directed graph and then do DFS.

TOPOLOGICAL SORT - DFS EXAMPLE

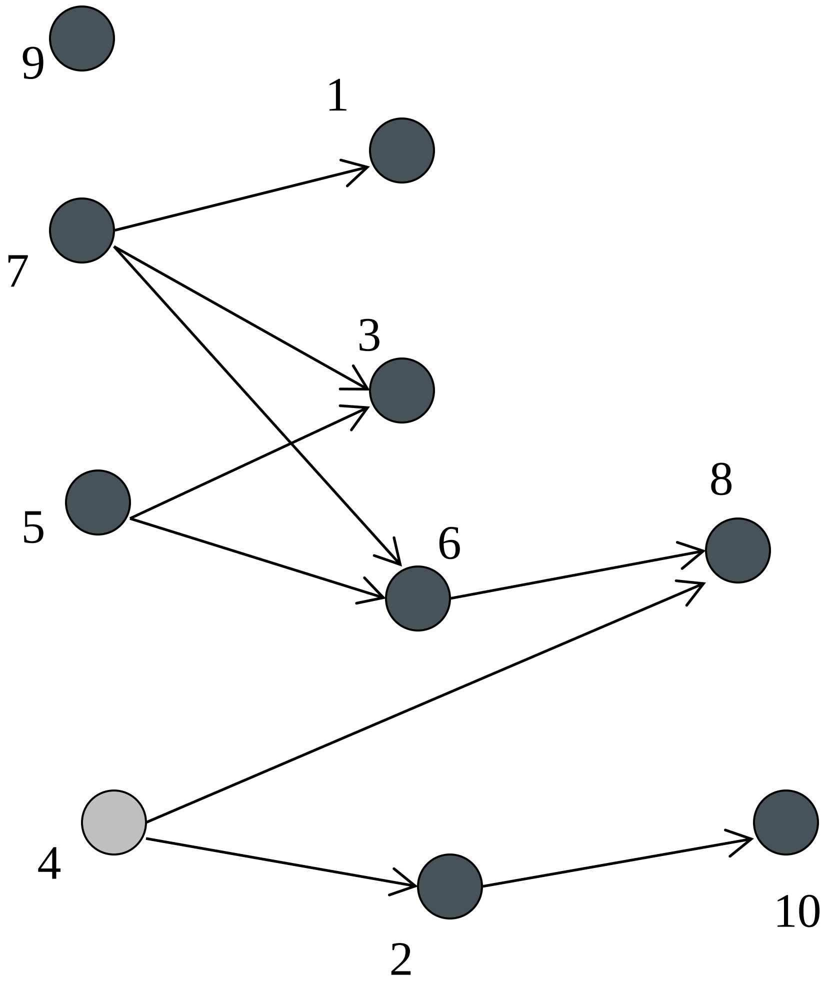


Tasks are shown as a directed graph.

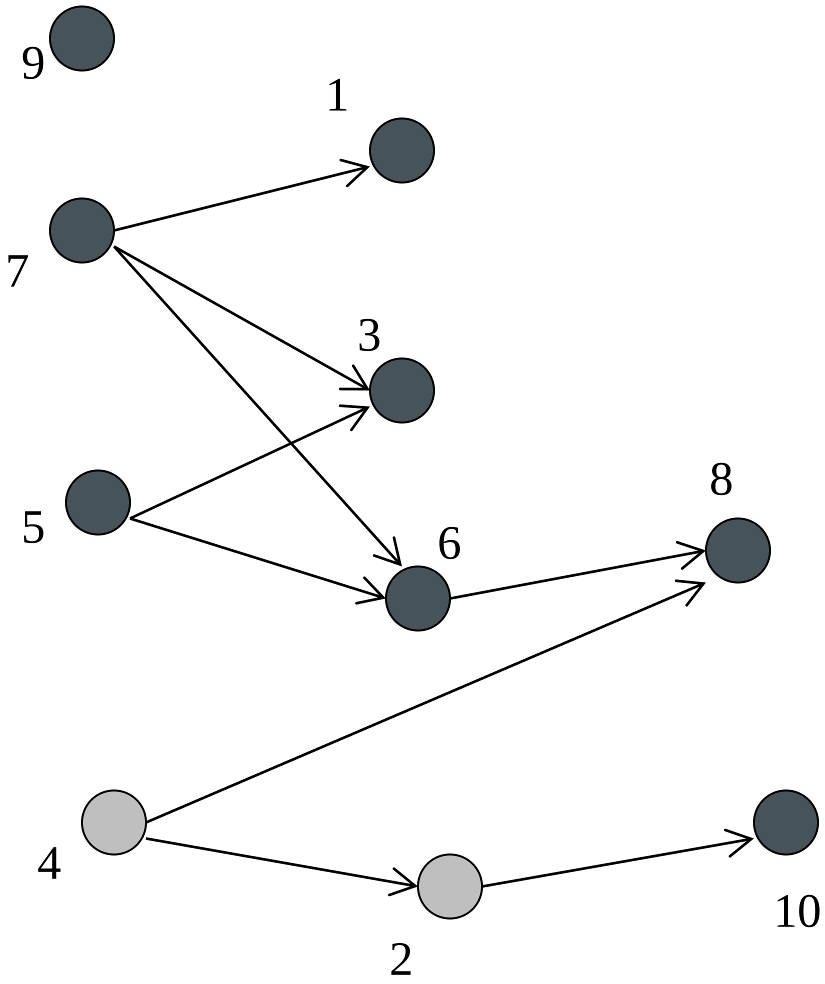
TOPOLOGICAL SORT - DFS EXAMPLE



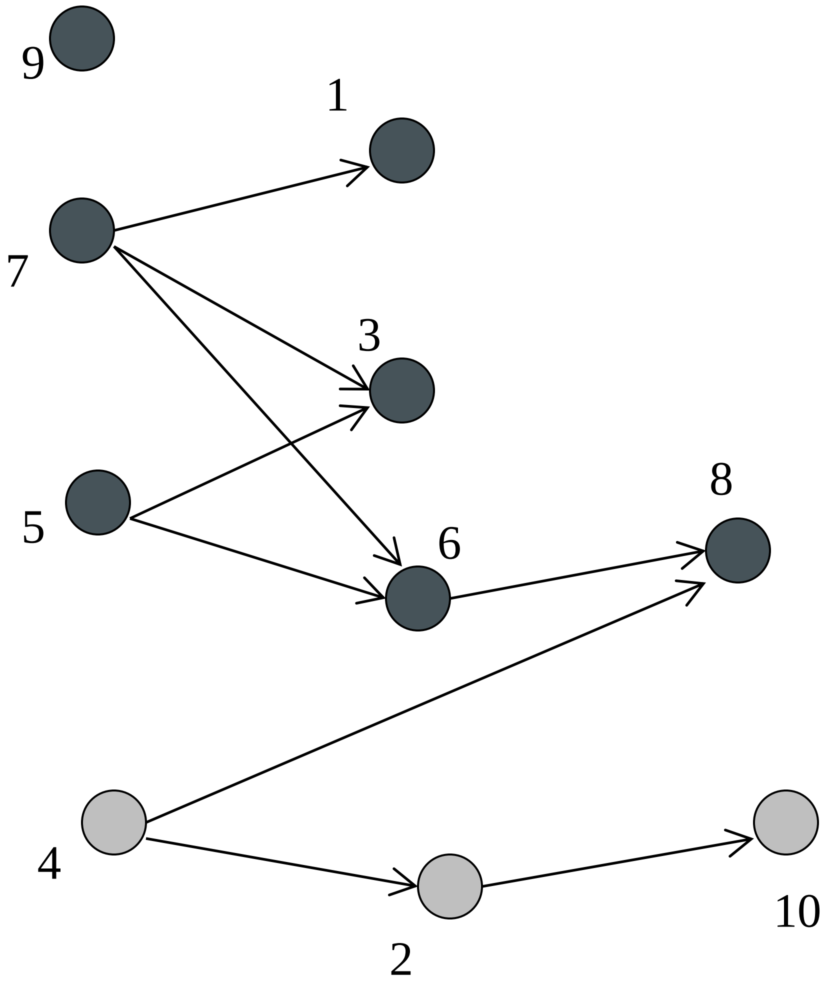
TOPOLOGICAL SORT - DFS EXAMPLE

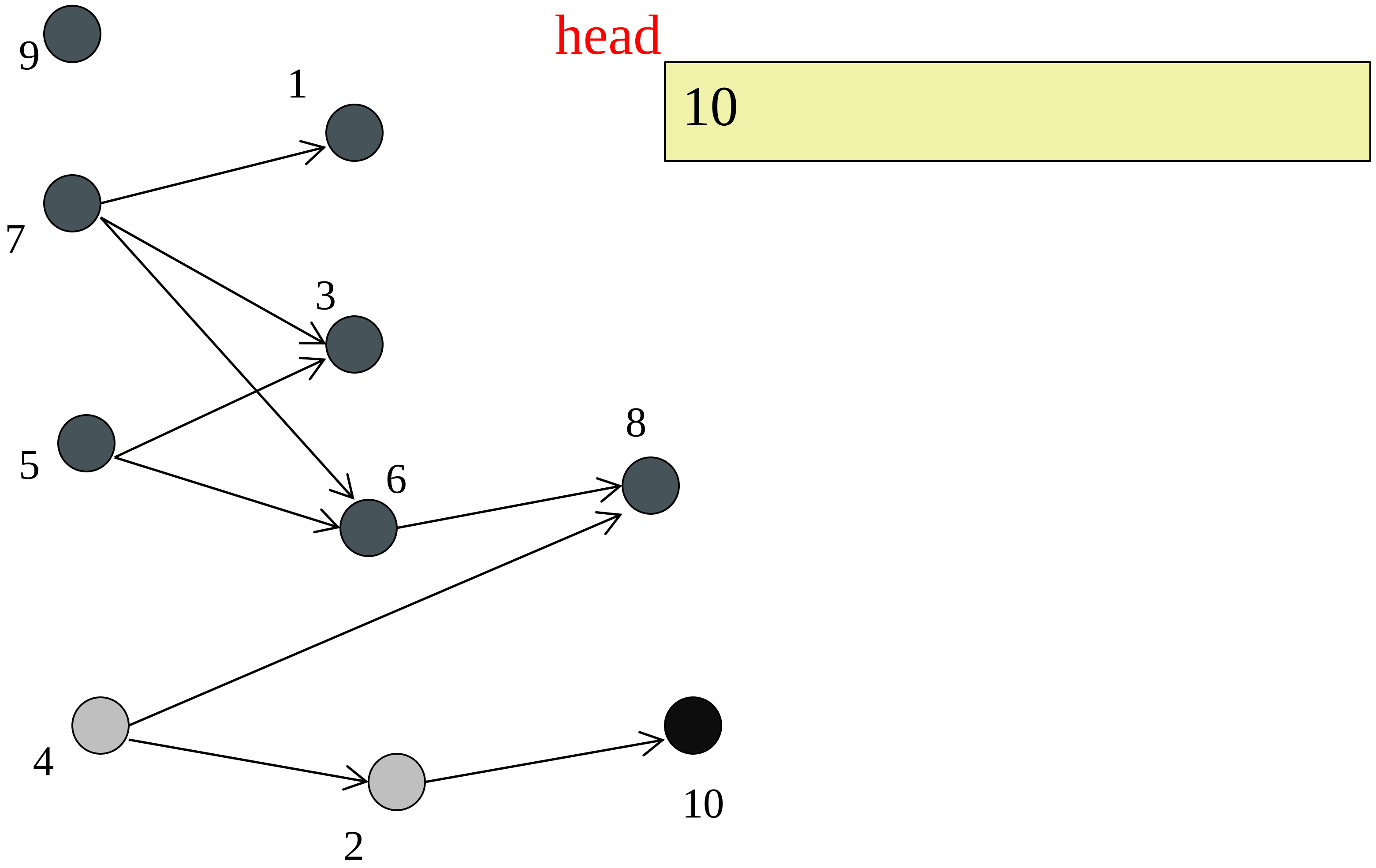


TOPOLOGICAL SORT - DFS EXAMPLE

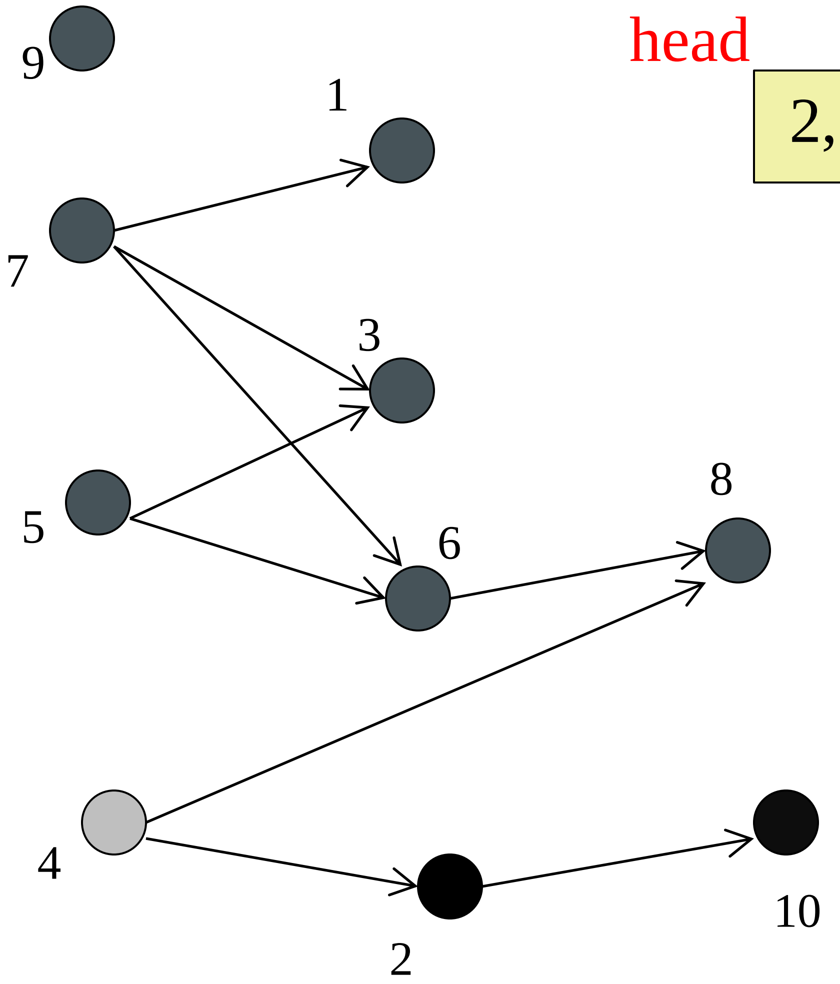


TOPOLOGICAL SORT - DFS EXAMPLE




9
1
7
3
5
6
8
4
2
10
head
10

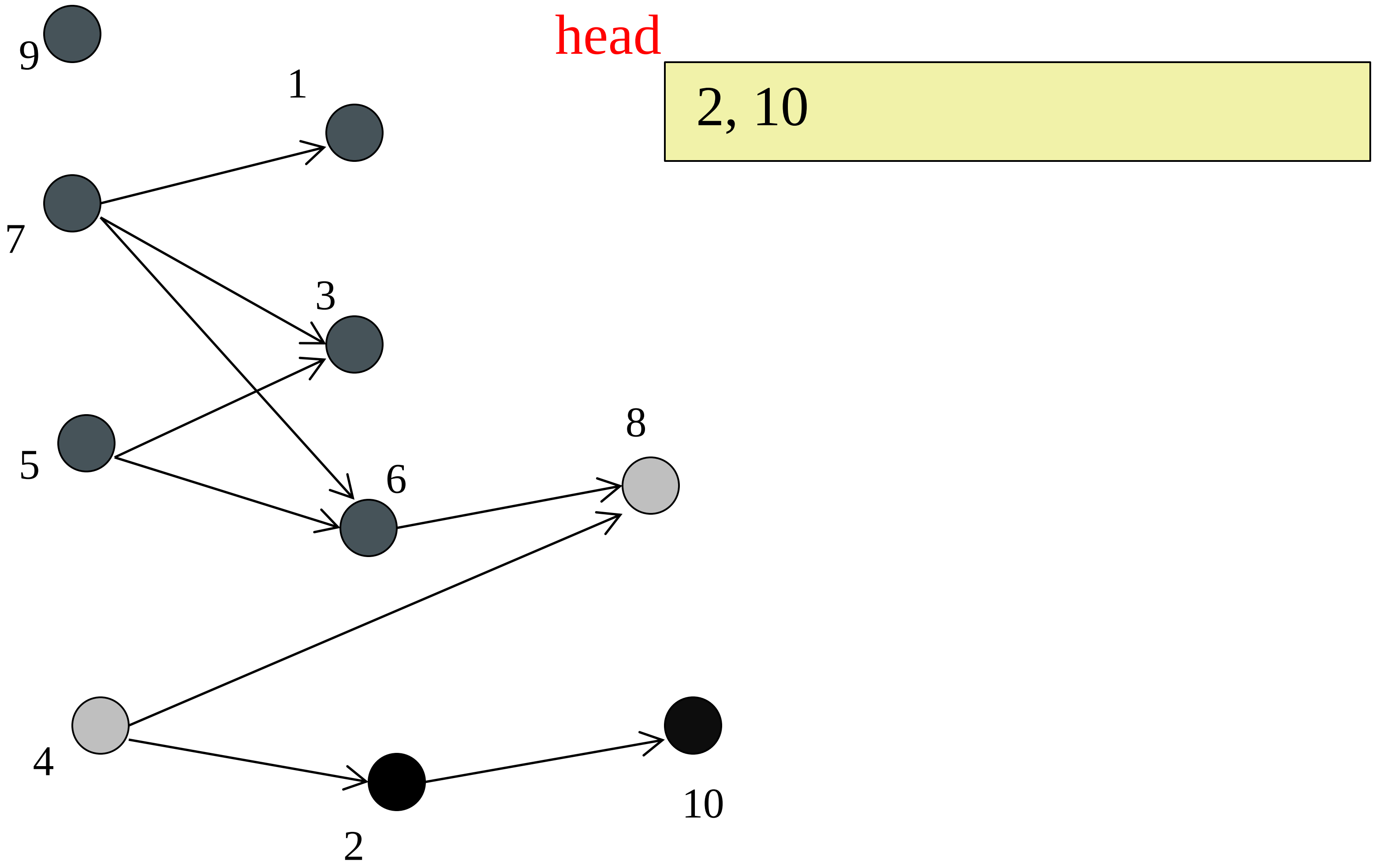
TOPOLOGICAL SORT - DFS EXAMPLE



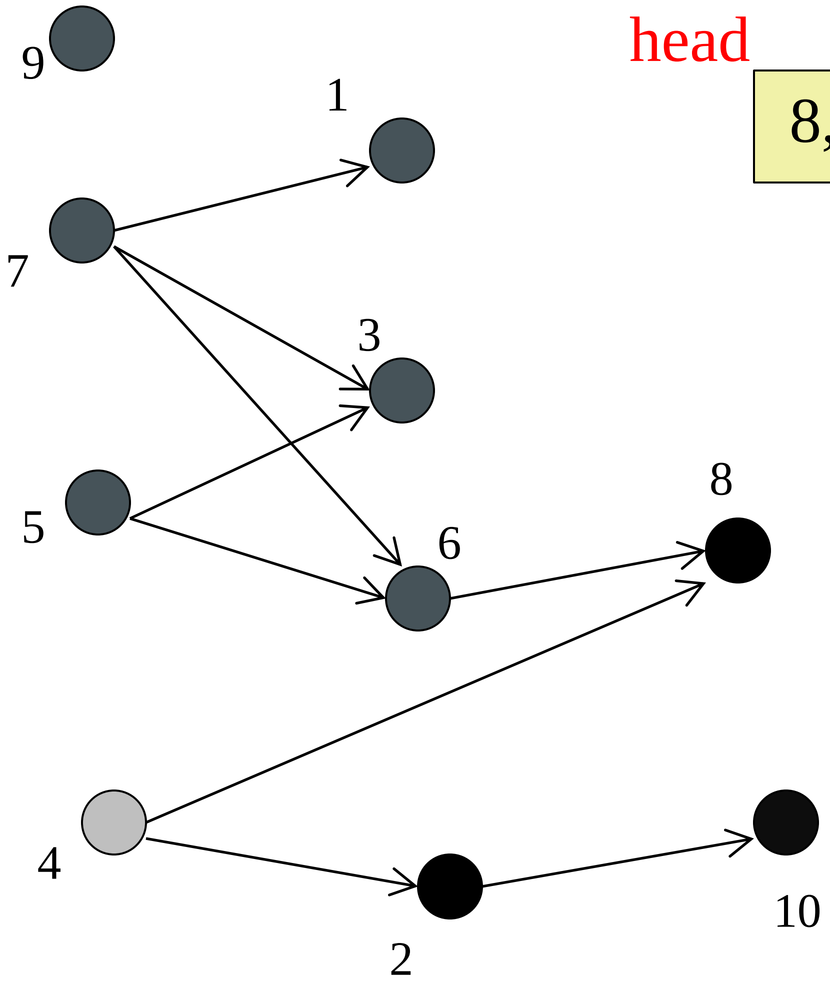
head

2, 10

TOPOLOGICAL SORT - DFS EXAMPLE



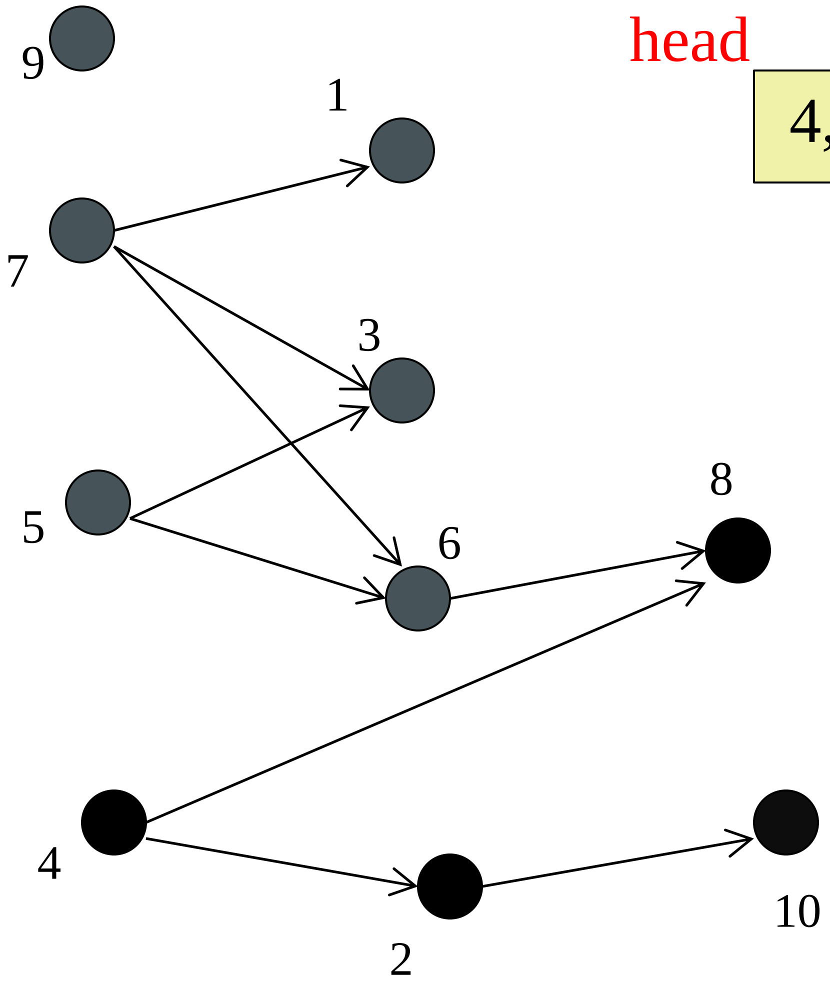
TOPOLOGICAL SORT - DFS EXAMPLE



head

8, 2, 10

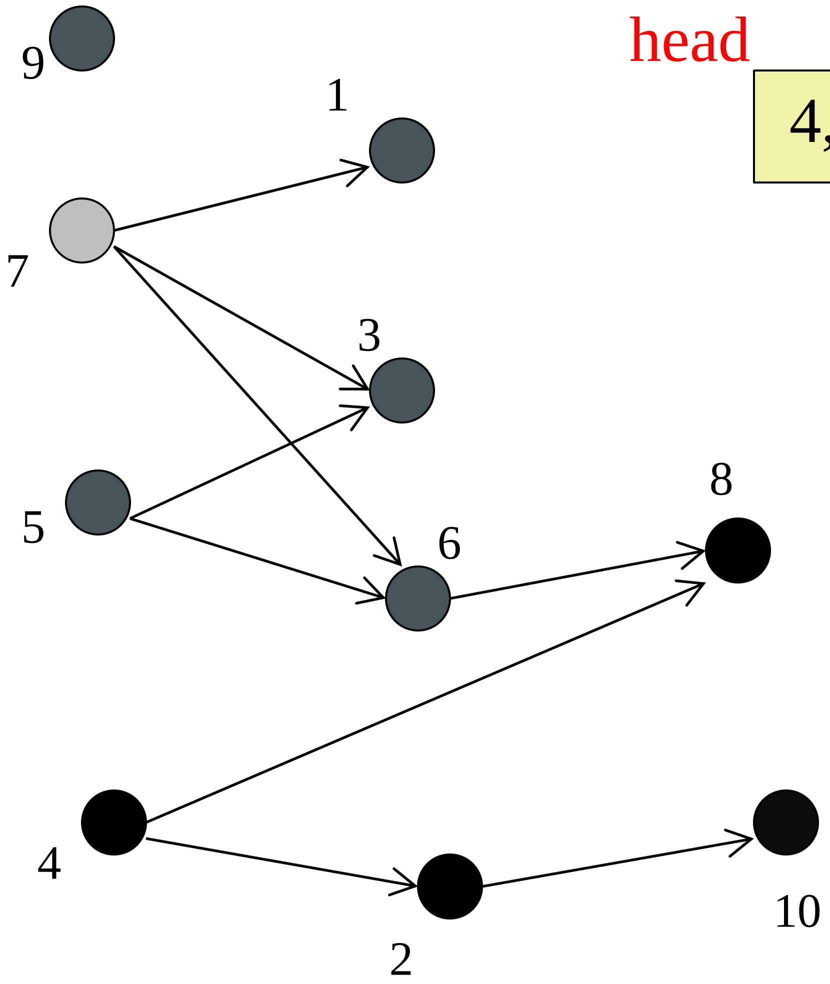
TOPOLOGICAL SORT - DFS EXAMPLE



head

4, 8, 2, 10

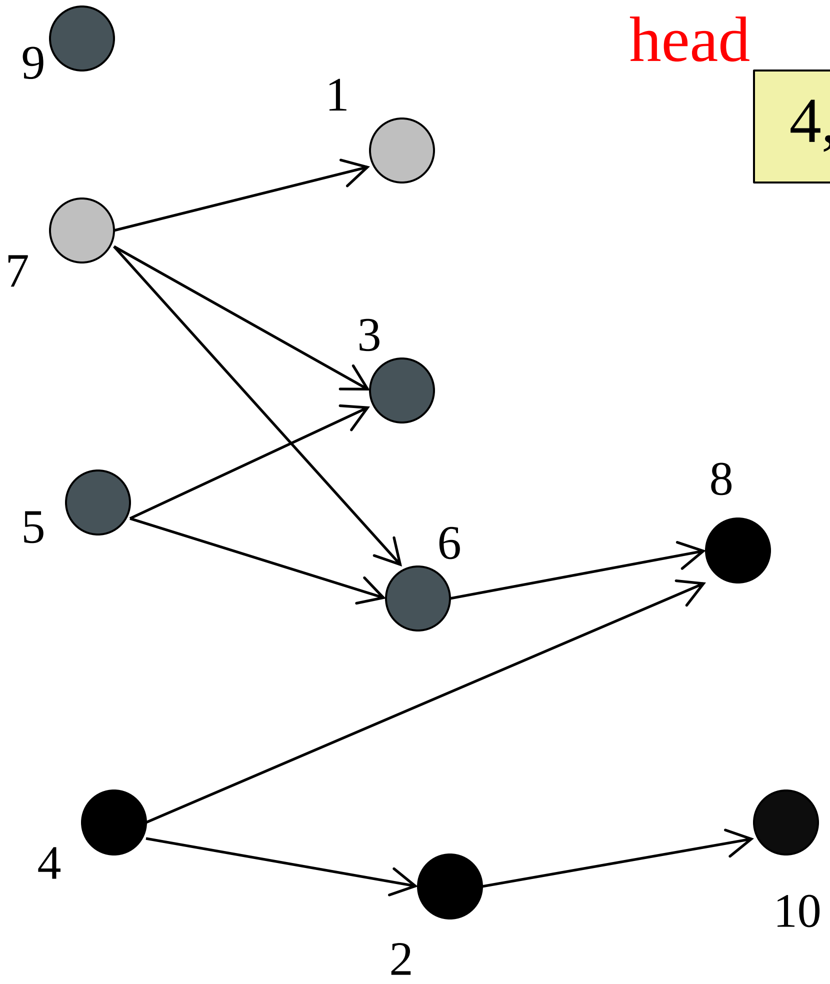
TOPOLOGICAL SORT - DFS EXAMPLE



head

4, 8, 2, 10

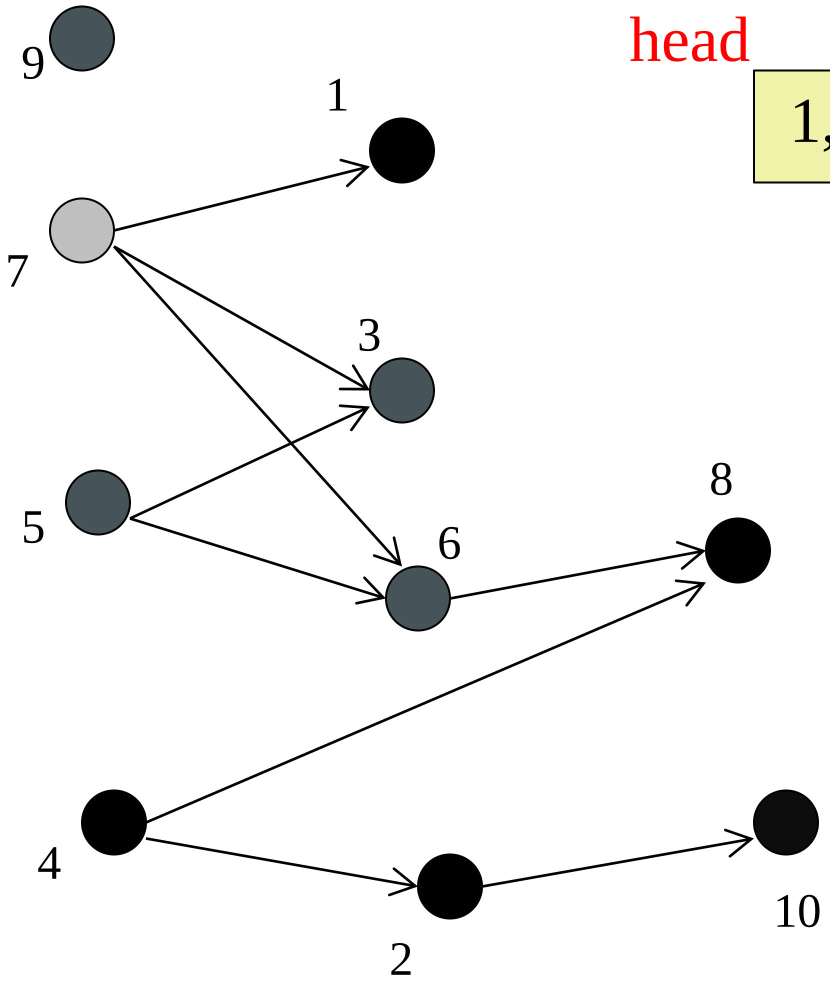
TOPOLOGICAL SORT - DFS EXAMPLE



head

4, 8, 2, 10

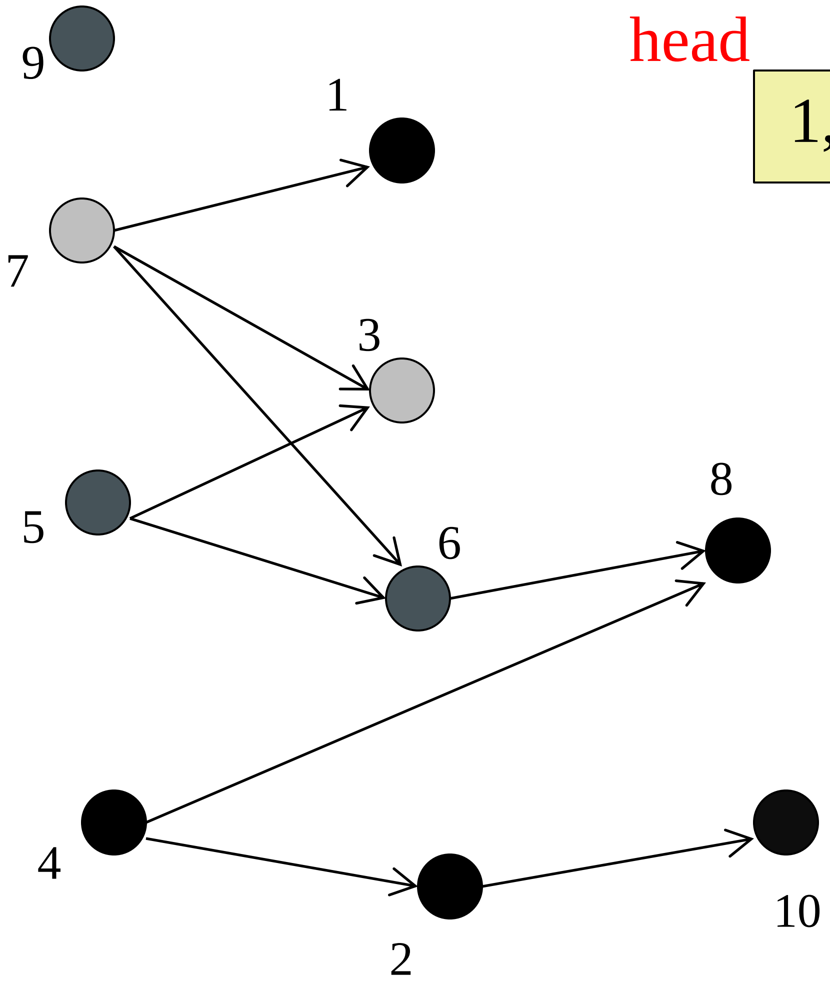
TOPOLOGICAL SORT - DFS EXAMPLE



head

1, 4, 8, 2, 10

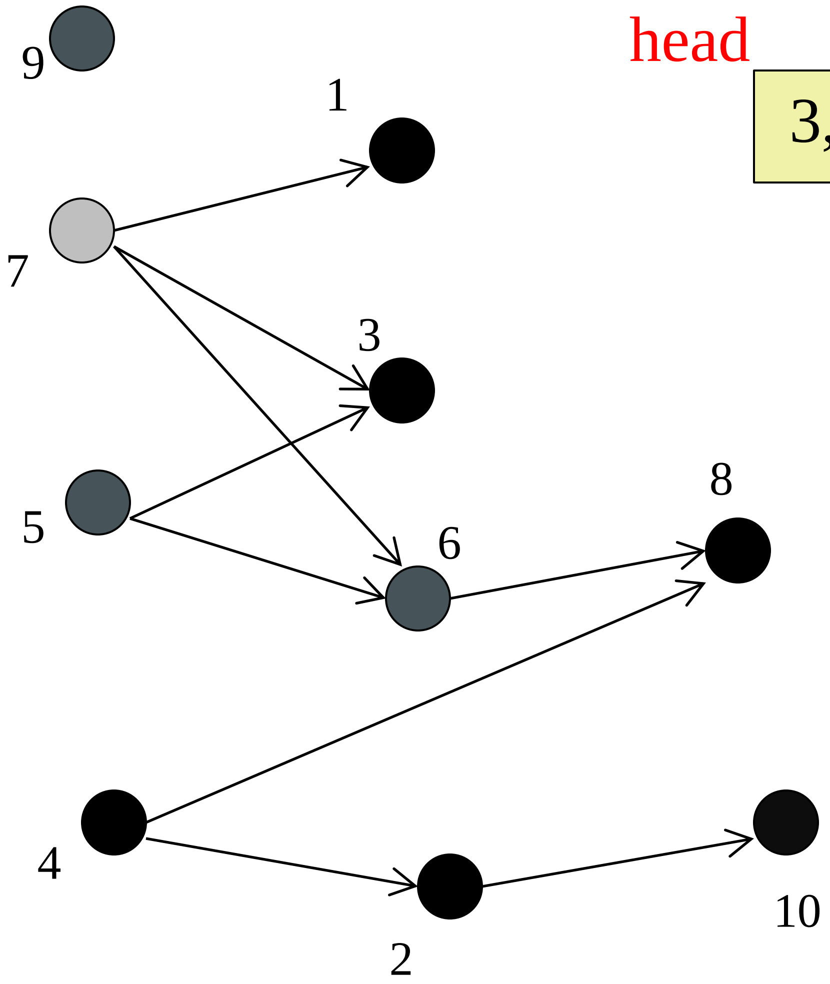
TOPOLOGICAL SORT - DFS EXAMPLE



head

1, 4, 8, 2, 10

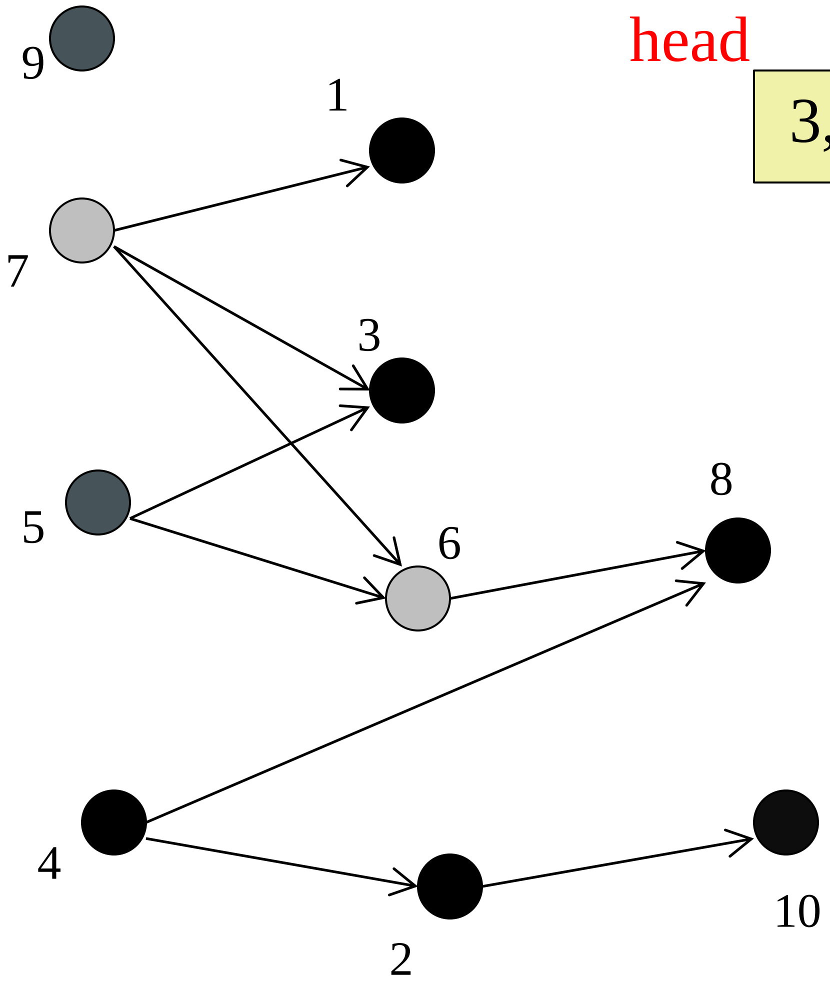
TOPOLOGICAL SORT - DFS EXAMPLE



head

3, 1, 4, 8, 2, 10

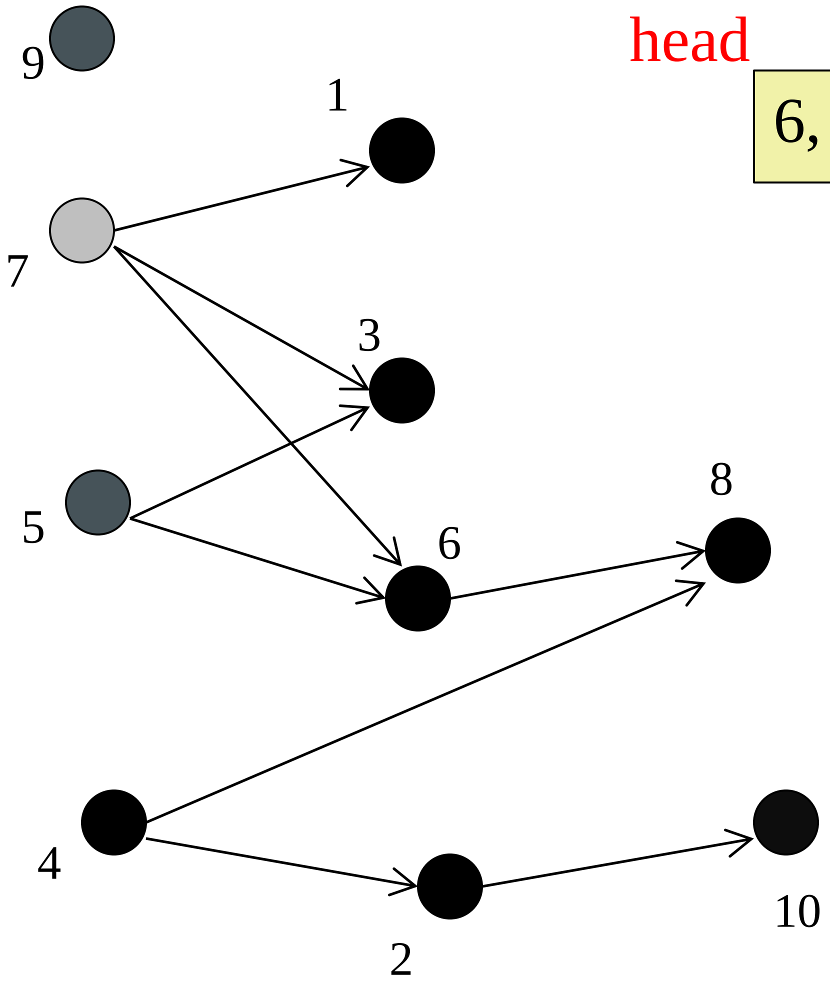
TOPOLOGICAL SORT - DFS EXAMPLE



head

3, 1, 4, 8, 2, 10

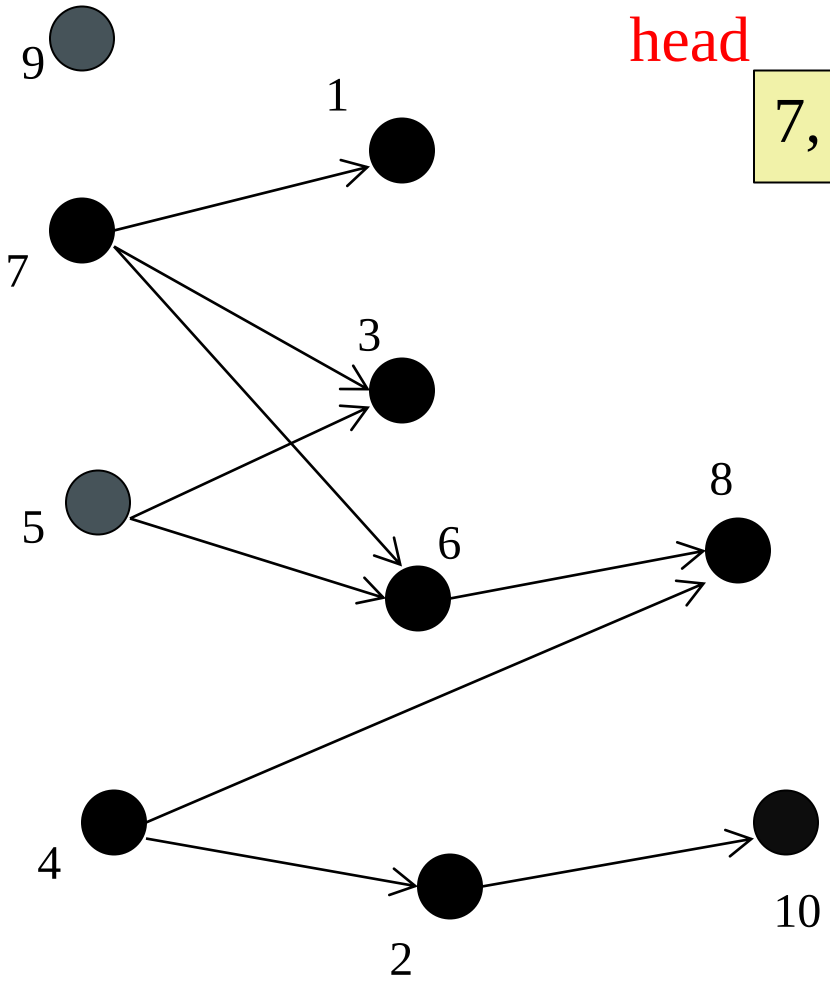
TOPOLOGICAL SORT - DFS EXAMPLE



head

6, 3, 1, 4, 8, 2, 10

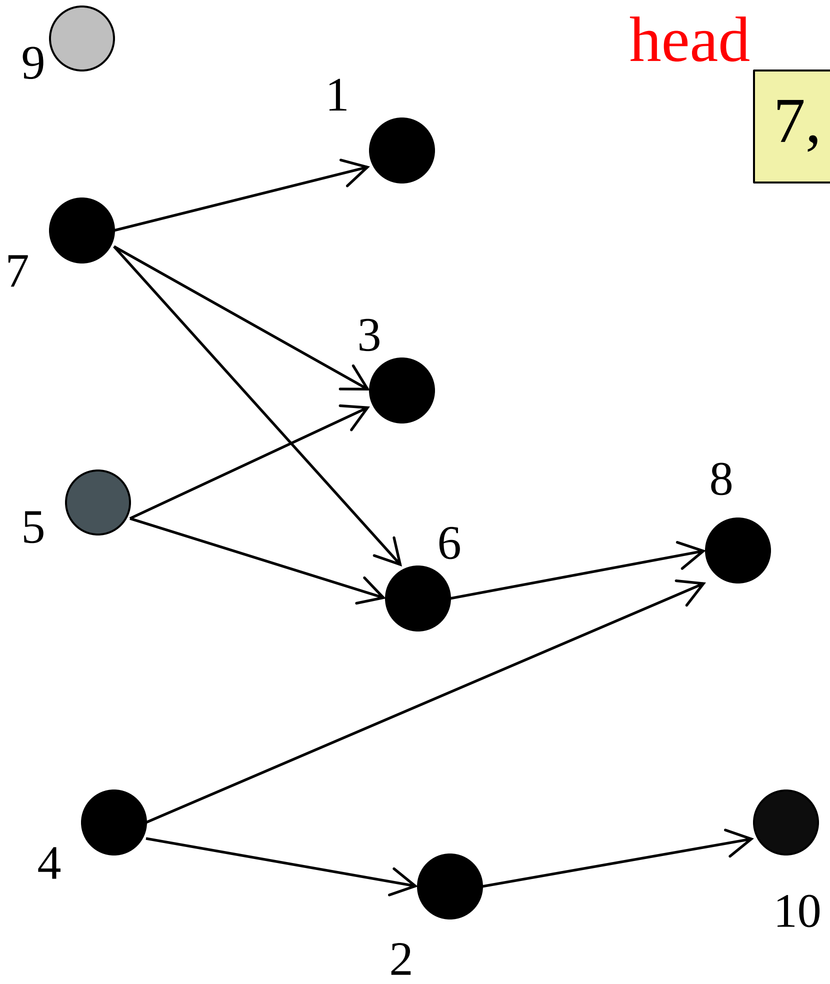
TOPOLOGICAL SORT - DFS EXAMPLE



head

7, 6, 3, 1, 4, 8, 2, 10

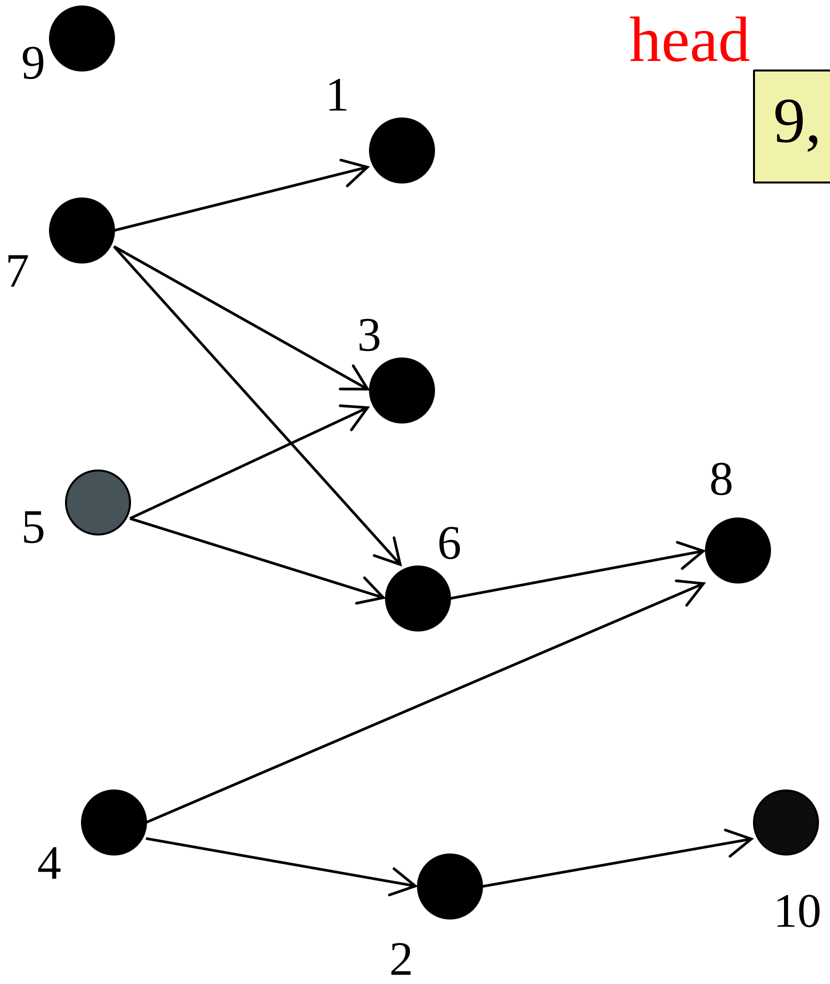
TOPOLOGICAL SORT - DFS EXAMPLE



head

7, 6, 3, 1, 4, 8, 2, 10

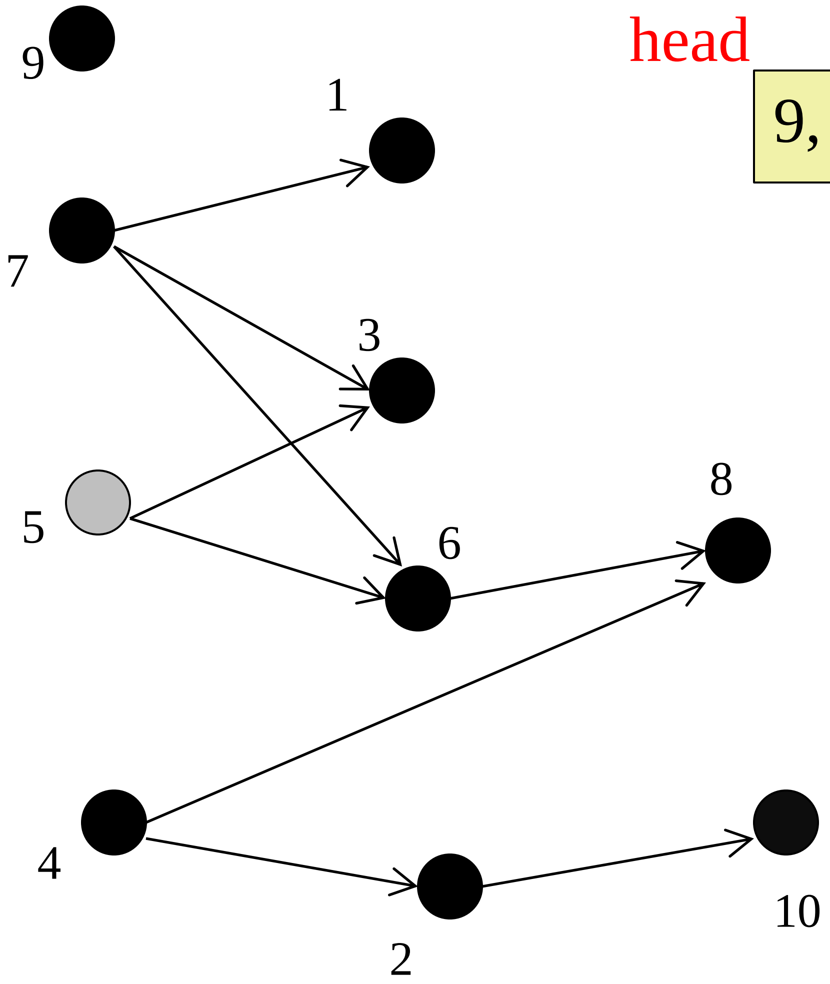
TOPOLOGICAL SORT - DFS EXAMPLE



head

9, 7, 6, 3, 1, 4, 8, 2, 10

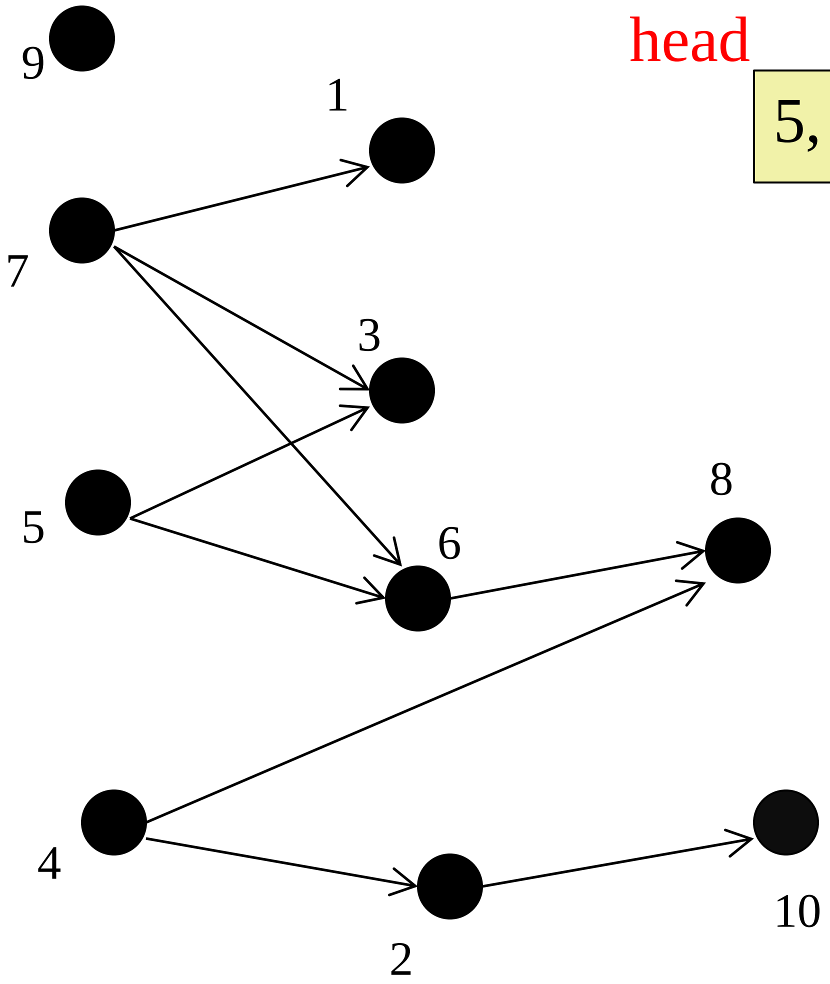
TOPOLOGICAL SORT - DFS EXAMPLE



head

9, 7, 6, 3, 1, 4, 8, 2, 10

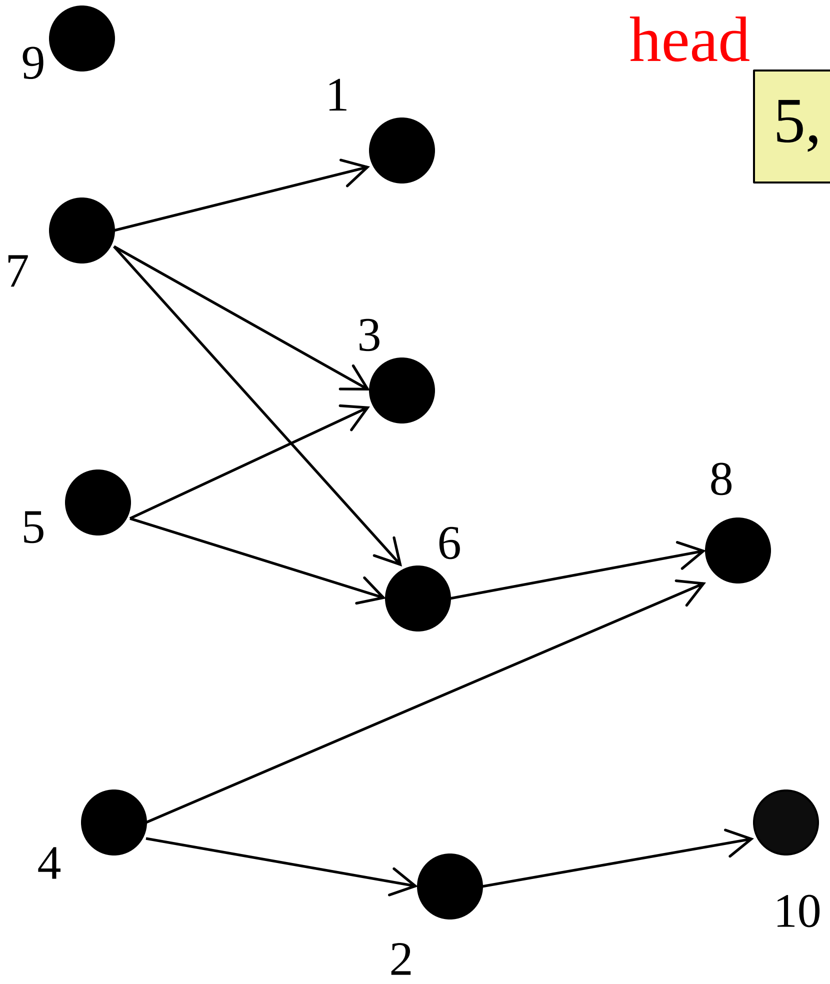
TOPOLOGICAL SORT - DFS EXAMPLE



head

5, 9, 7, 6, 3, 1, 4, 8, 2, 10

TOPOLOGICAL SORT - DFS EXAMPLE



head

5, 9, 7, 6, 3, 1, 4, 8, 2, 10

The final order or jobs

Time complexity = DFS complexity $\Rightarrow O(V + E)$

There can be many orders that meet the requirements

TOPOLOGICAL SORT - DFS COMPLEXITY

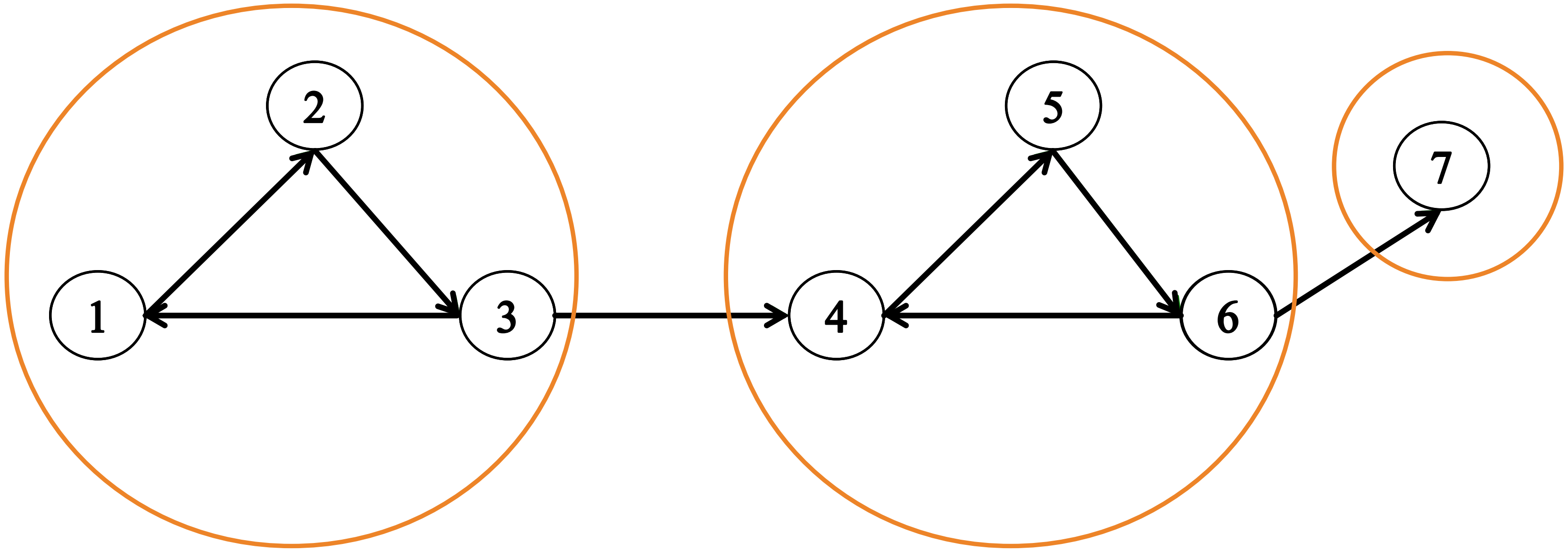
- Time complexity: $O(V + E)$

You traverse each edge once and you check each vertex once

GRAPH ALGORITHMS. STRONGLY CONNECTED COMPONENTS

STRONGLY CONNECTED COMPONENTS

- A graph is said to be strongly connected if every vertex is reachable from every other vertex.
- The strongly connected components of an arbitrarily directed graph form a partition into subgraphs that are themselves strongly connected.
- A subset of a directed graph is called strongly connected if there is a path in each direction between each pair of vertices of the subset.



PATH-BASED DEPTH-FIRST SEARCH ALGORITHM

Algorithm 1 Path-Based Depth-First Search Algorithm

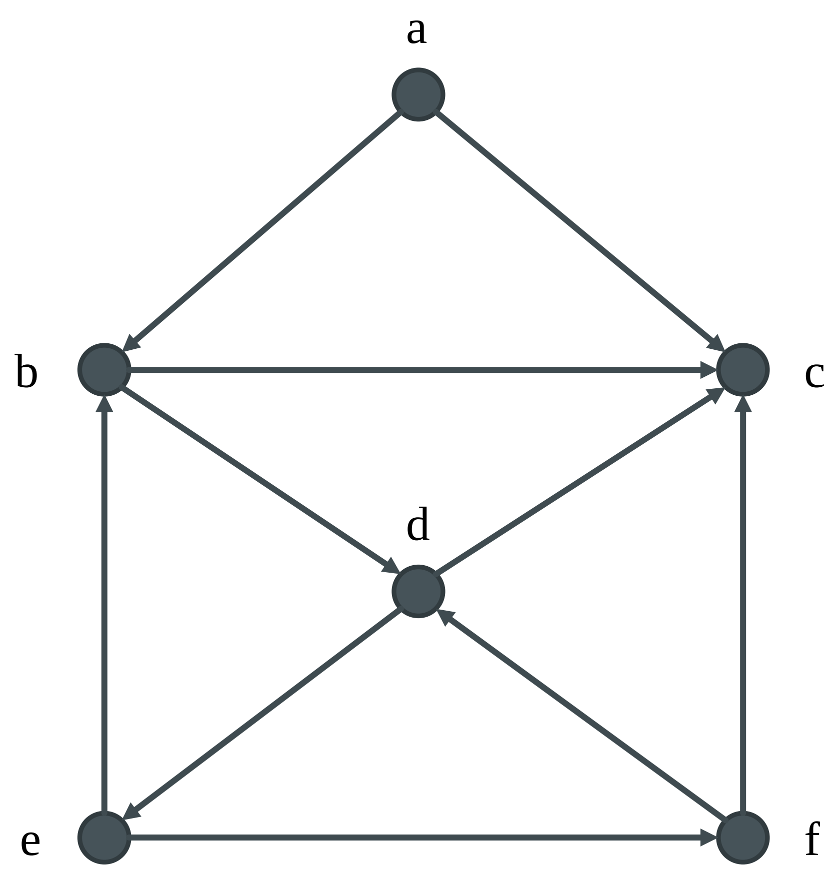
- **Input:** A directed graph $G = (V, L)$
- **Output:** Set of SCCs represented by SCC (initially empty)
- **for** $v \in V$ **do**
    - Set the node as *unseen*.
- Initiate a path $P = \{\}$
- **for** $v \in V$ **do**
    - **if** v is *unseen* **then**
        - DFS(v).

Algorithm 2 DFS(v)

- **if** v has no outgoing arc **then**
    - Consider v as an SCC and add it to SCC
    - Set v as *seen*.
    - Remove it from P.
- **else**
    - **for** arc $(v, w) \in E$ **do**
        - **if** $w \in P$ $(v_i = w)$ **then**
            - Contract the nodes v_i, v_{i+1}, ... , $v_{|P|}$.
        - **else**
            - Add w to P.
            - Set w as *seen*.
            - DFS(w).

PATH-BASED DEPTH-FIRST SEARCH ALGORITHM

SCCs **P**



Algorithm 1 Path-Based Depth-First Search Algorithm

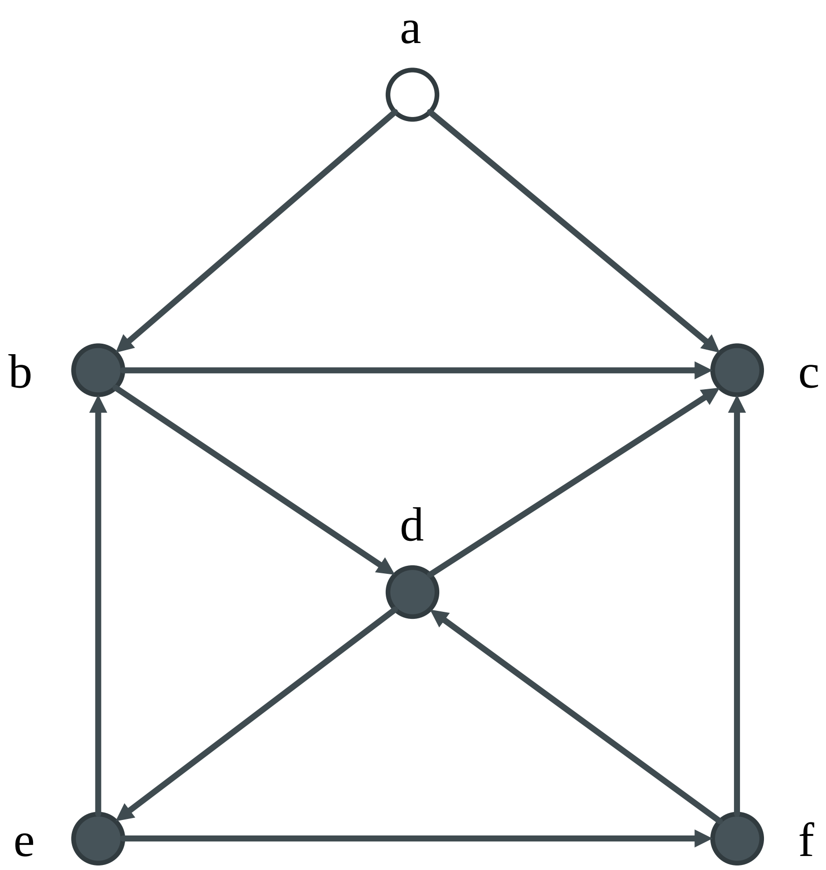
Input: A directed graph $G = (V, L)$
Output: Set of SCCs represented by SCC (initially empty)
for $v \in V$ **do**
  Set the node as *unseen*.
Initiate a path $P = \{\}$
for $v \in V$ **do**
  if v is *unseen* **then**
    DFS(v).

Algorithm 2 DFS(v)

if v has no outgoing arc **then**
  Consider v as an SCC and add it to SCC
  Set v as *seen*.
  Remove it from P.
else
  for arc $(v, w) \in E$ **do**
    if $w \in P$ $(v_i = w)$ **then**
      Contract the nodes v_i, v_{i+1}, ... , $v_{|P|}$.
    else
      Add w to P.
      Set w as *seen*.
      DFS(w).

PATH-BASED DEPTH-FIRST SEARCH ALGORITHM

SCCs



P

Algorithm 1 Path-Based Depth-First Search Algorithm

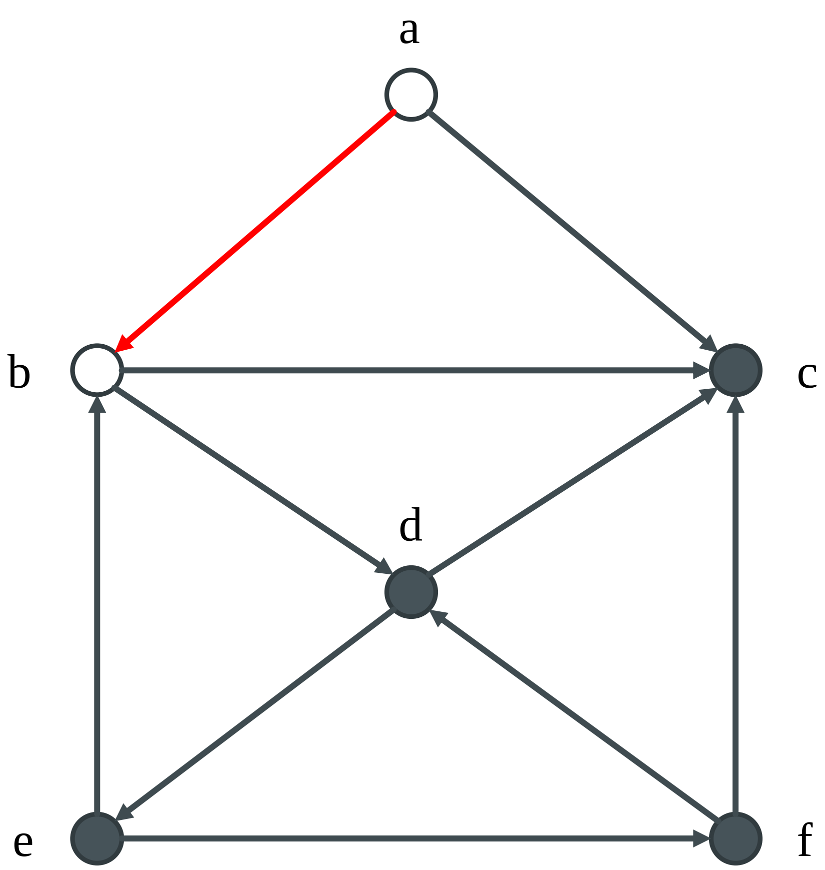
Input: A directed graph $G = (V, L)$
Output: Set of SCCs represented by SCC (initially empty)
for $v \in V$ **do**
    Set the node as *unseen*.
Initiate a path $P = \{\}$
for $v \in V$ **do**
    if v is *unseen* **then**
        DFS(v).

Algorithm 2 DFS(v)

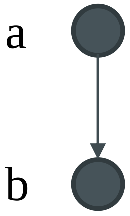
if v has no outgoing arc **then**
    Consider v as an SCC and add it to SCC
    Set v as *seen*.
    Remove it from P.
else
    for arc $(v, w) \in E$ **do**
        if $w \in P$ $(v_i = w)$ **then**
            Contract the nodes v_i, v_{i+1}, ... , $v_{|P|}$.
        else
            Add w to P.
            Set w as *seen*.
            DFS(w).

PATH-BASED DEPTH-FIRST SEARCH ALGORITHM

SCCs



P



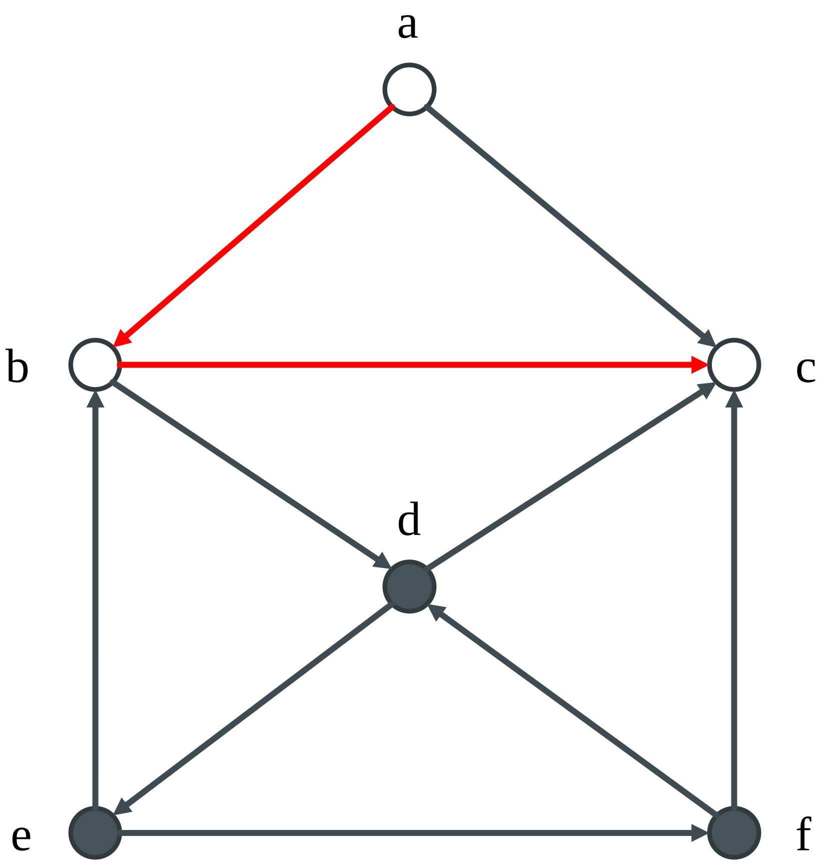
Algorithm 1 Path-Based Depth-First Search Algorithm

```
Input: A directed graph G = (V, L)
Output: Set of SCCs represented by SCC (initially empty)
for v ∈ V do
    Set the node as unseen.
Initiate a path P = {}
for v ∈ V do
    if v is unseen then
        DFS(v).
```

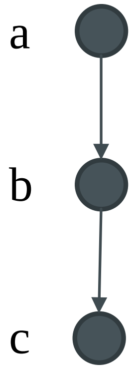
Algorithm 2 DFS(v)

```
if v has no outgoing arc then
    Consider v as an SCC and add it to SCC
    Set v as seen.
    Remove it from P.
else
    for arc (v, w) ∈ E do
        if w ∈ P (v_i = w) then
            Contract the nodes v_i, v_{i+1}, ... , v_{|P|}.
        else
            Add w to P.
            Set w as seen.
            DFS(w).
```

SCCs



P



Algorithm 1 Path-Based Depth-First Search Algorithm

Input: A directed graph $G = (V, L)$
Output: Set of SCCs represented by SCC (initially empty)
for $v \in V$ **do**
    Set the node as *unseen*.
Initiate a path $P = \{\}$
for $v \in V$ **do**
    if v is *unseen* **then**
        DFS(v).

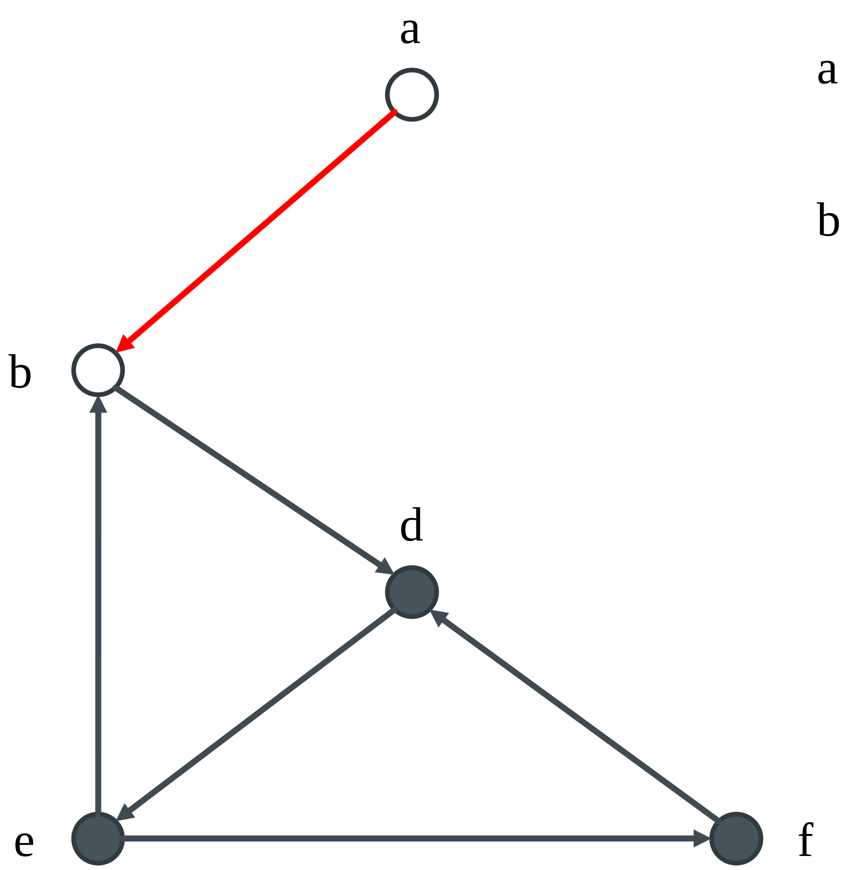
Algorithm 2 DFS(v)

if v has no outgoing arc **then**
    Consider v as an SCC and add it to SCC
    Set v as *seen*.
    Remove it from P.
else
    for arc $(v, w) \in E$ **do**
        if $w \in P$ $(v_i = w)$ **then**
            Contract the nodes v_i, v_{i+1}, ... , $v_{|P|}$.
        else
            Add w to P.
            Set w as *seen*.
            DFS(w).

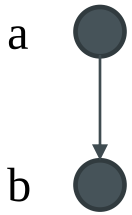
PATH-BASED DEPTH-FIRST SEARCH ALGORITHM

SCCs

{c}



P



Algorithm 1 Path-Based Depth-First Search Algorithm

Input: A directed graph $G = (V, L)$
Output: Set of SCCs represented by SCC (initially empty)
for $v \in V$ **do**
    Set the node as *unseen*.
Initiate a path $P = \{\}$
for $v \in V$ **do**
    if v is *unseen* **then**
        DFS(v).

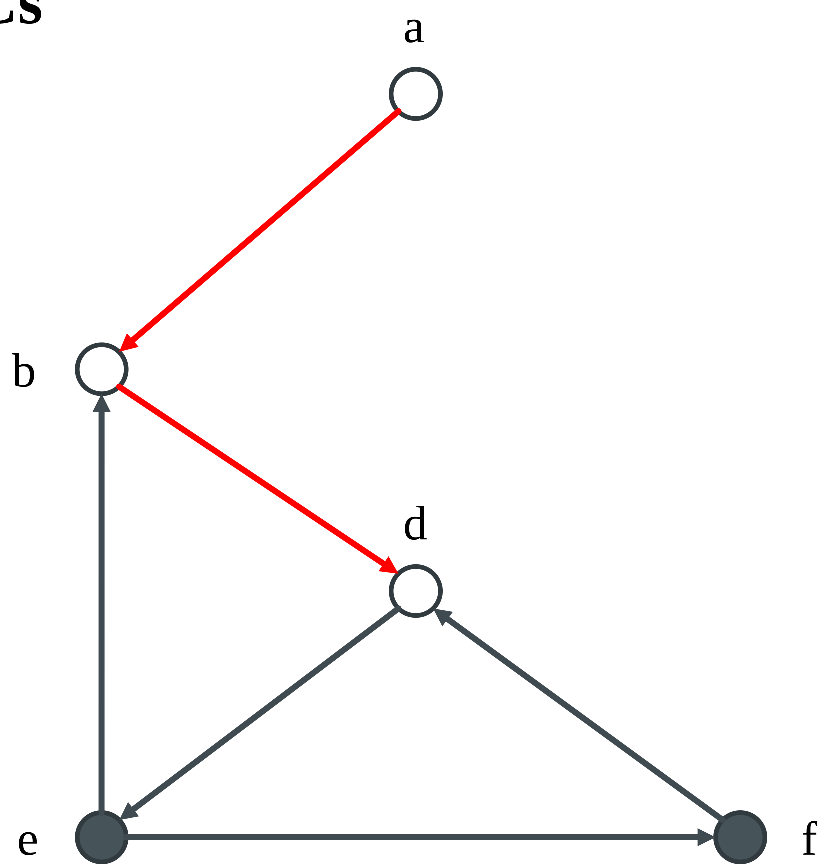
Algorithm 2 DFS(v)

if v has no outgoing arc **then**
    Consider v as an SCC and add it to SCC
    Set v as *seen*.
    Remove it from P.
else
    for arc $(v, w) \in E$ **do**
        if $w \in P$ $(v_i = w)$ **then**
            Contract the nodes v_i, v_{i+1}, ... , $v_{|P|}$.
        else
            Add w to P.
            Set w as *seen*.
            DFS(w).

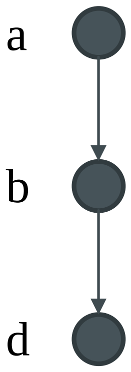
PATH-BASED DEPTH-FIRST SEARCH ALGORITHM

SCCs

{c}



P



Algorithm 1 Path-Based Depth-First Search Algorithm

Input: A directed graph $G = (V, L)$
Output: Set of SCCs represented by SCC (initially empty)
for $v \in V$ **do**
    Set the node as *unseen*.
Initiate a path $P = \{\}$
for $v \in V$ **do**
    if v is *unseen* **then**
        DFS(v).

Algorithm 2 DFS(v)

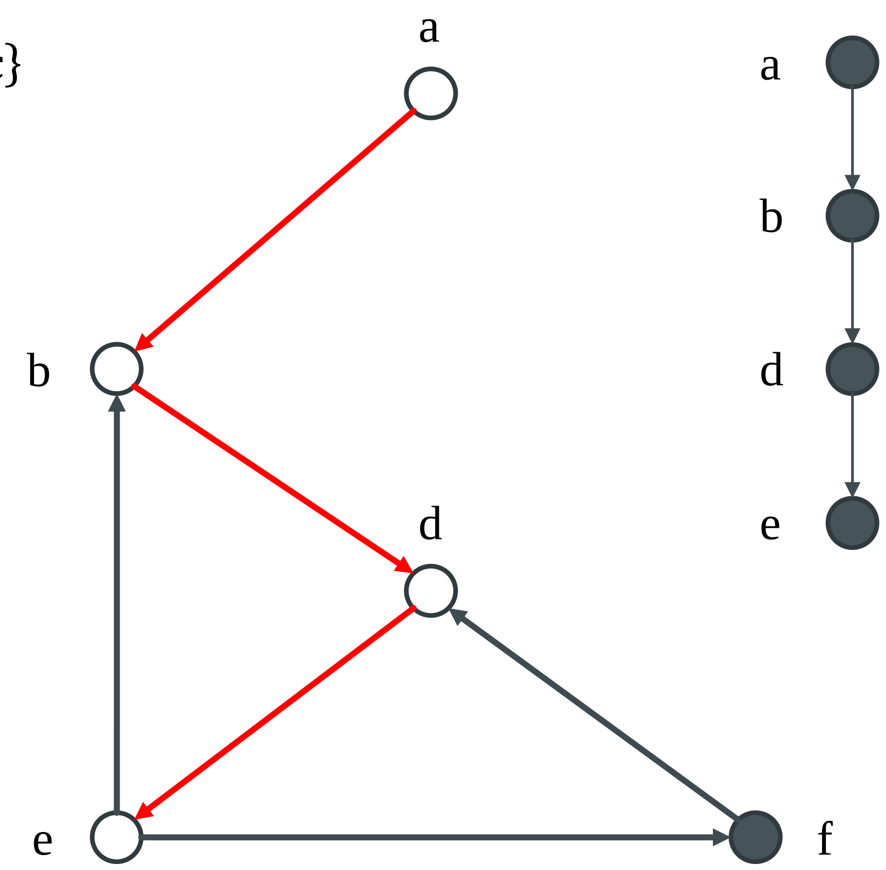
---

if v has no outgoing arc **then**
    Consider v as an SCC and add it to SCC
    Set v as *seen*.
    Remove it from P.
else
    for arc $(v, w) \in E$ **do**
        if $w \in P$ $(v_i = w)$ **then**
            Contract the nodes v_i, v_{i+1}, ... , $v_{|P|}$.
        else
            Add w to P.
            Set w as *seen*.
            DFS(w).

PATH-BASED DEPTH-FIRST SEARCH ALGORITHM

SCCs

{c}



Algorithm 1 Path-Based Depth-First Search Algorithm

Input: A directed graph $G = (V, L)$
Output: Set of SCCs represented by *SCC* (initially empty)
for $v \in V$ **do**
    Set the node as *unseen*.
Initiate a path $P = \{\}$
for $v \in V$ **do**
    if v is *unseen* **then**
        DFS(v).

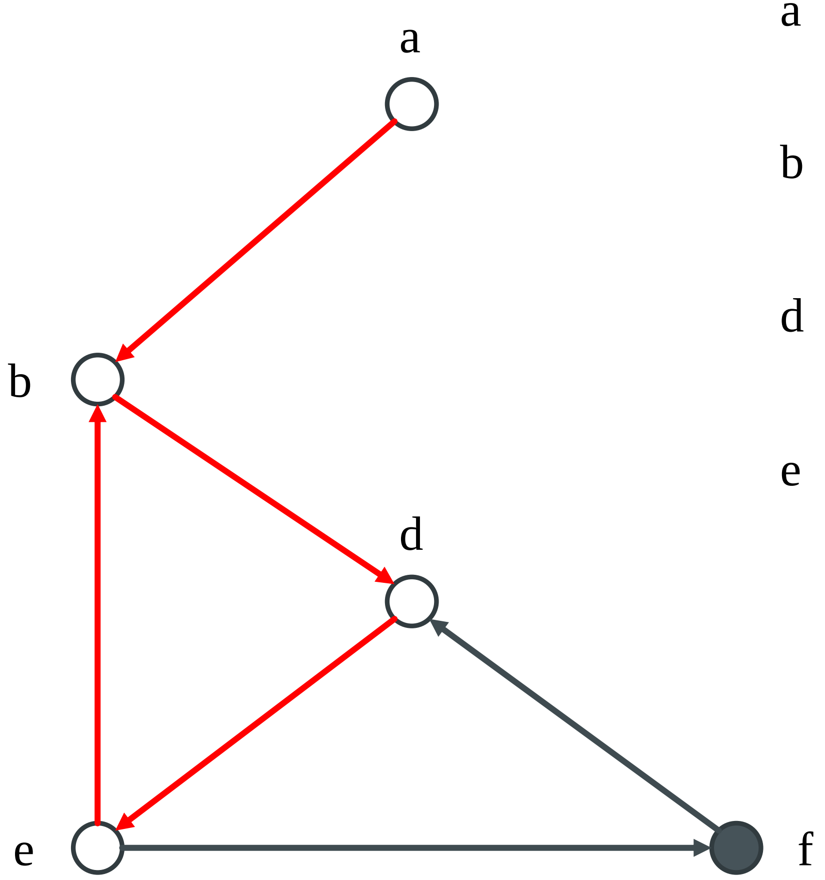
Algorithm 2 DFS(v)

if v has no outgoing arc **then**
    Consider v as an SCC and add it to *SCC*
    Set v as *seen*.
    Remove it from P.
else
    for arc $(v, w) \in E$ **do**
        if $w \in P$ $(v_i = w)$ **then**
            Contract the nodes v_i, v_{i+1}, ... , $v_{|P|}$.
        else
            Add w to P.
            Set w as *seen*.
            DFS(w).

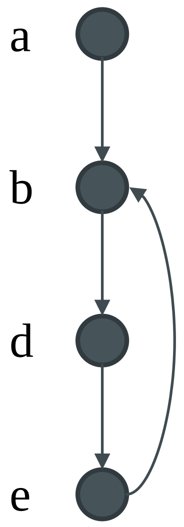
PATH-BASED DEPTH-FIRST SEARCH ALGORITHM

SCCs

{c}



P



Algorithm 1 Path-Based Depth-First Search Algorithm

Input: A directed graph $G = (V, L)$
Output: Set of SCCs represented by SCC (initially empty)
for $v \in V$ **do**
    Set the node as *unseen*.
Initiate a path $P = \{\}$
for $v \in V$ **do**
    if v is *unseen* **then**
        DFS(v).

Algorithm 2 DFS(v)

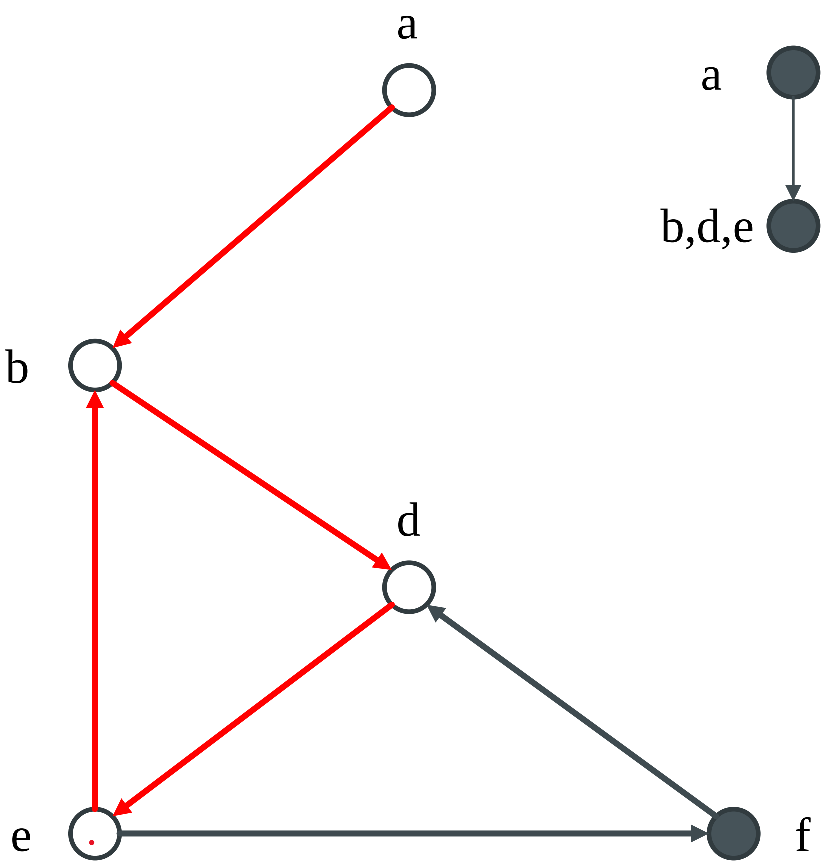
if v has no outgoing arc **then**
    Consider v as an SCC and add it to SCC
    Set v as *seen*.
    Remove it from P.
else
    for arc $(v, w) \in E$ **do**
        if $w \in P$ $(v_i = w)$ **then**
            Contract the nodes v_i, v_{i+1}, ... , $v_{|P|}$.
        else
            Add w to P.
            Set w as *seen*.
            DFS(w).

PATH-BASED DEPTH-FIRST SEARCH ALGORITHM

SCCs

{c}

P



Algorithm 1 Path-Based Depth-First Search Algorithm

```
Input: A directed graph G = (V, L)
Output: Set of SCCs represented by SCC (initially empty)
for v ∈ V do
    Set the node as unseen.
Initiate a path P = {}
for v ∈ V do
    if v is unseen then
        DFS(v).
```

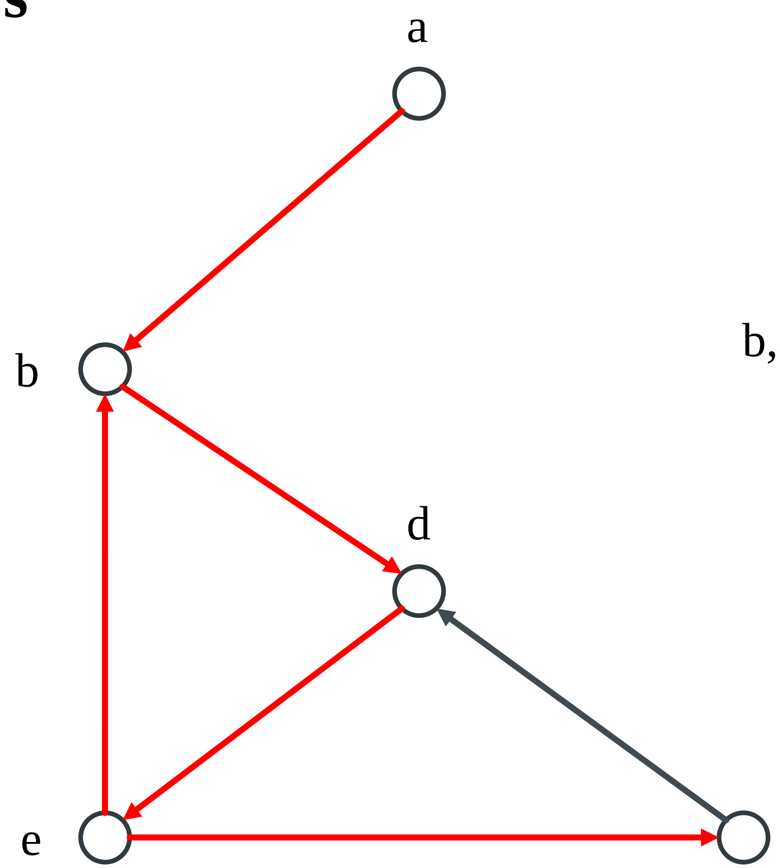
Algorithm 2 DFS(v)

```
if v has no outgoing arc then
    Consider v as an SCC and add it to SCC
    Set v as seen.
    Remove it from P.
else
    for arc (v, w) ∈ E do
        if w ∈ P (v_i = w) then
            Contract the nodes v_i, v_{i+1}, ... , v_{|P|}.
        else
            Add w to P.
            Set w as seen.
            DFS(w).
```

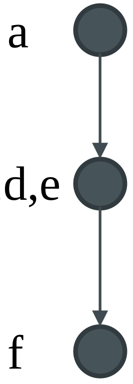
PATH-BASED DEPTH-FIRST SEARCH ALGORITHM

SCCs

{c}



P



Algorithm 1 Path-Based Depth-First Search Algorithm

Input: A directed graph $G = (V, L)$
Output: Set of SCCs represented by SCC (initially empty)
for $v \in V$ **do**
    Set the node as *unseen*.
Initiate a path $P = \{\}$
for $v \in V$ **do**
    if v is *unseen* **then**
        DFS(v).

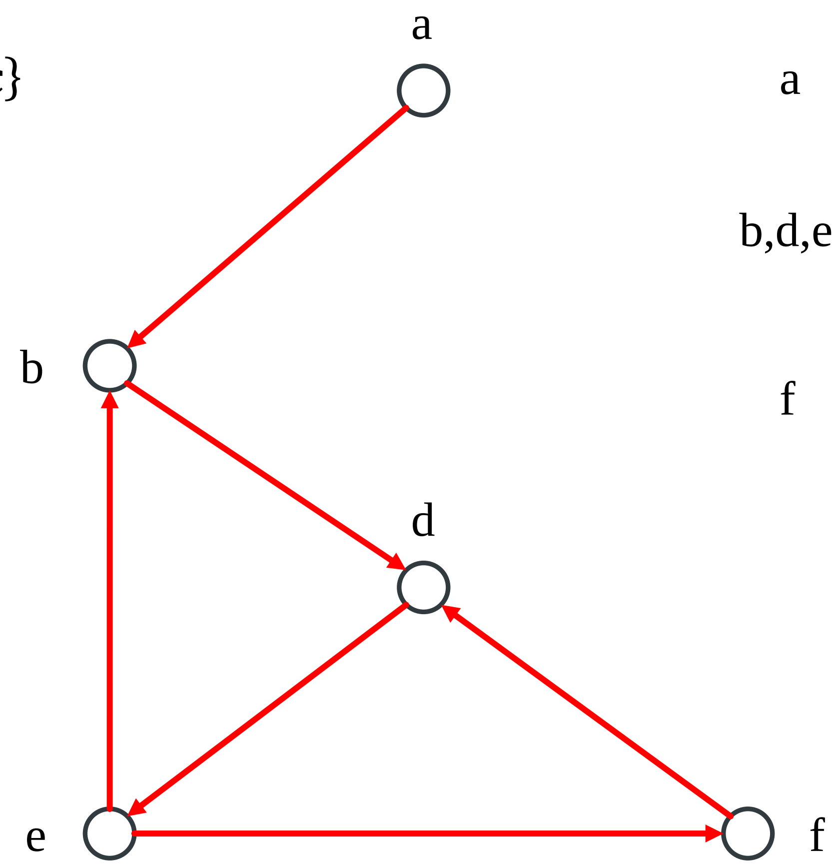
Algorithm 2 DFS(v)

if v has no outgoing arc **then**
    Consider v as an SCC and add it to SCC
    Set v as *seen*.
    Remove it from P.
else
    for arc $(v, w) \in E$ **do**
        if $w \in P$ $(v_i = w)$ **then**
            Contract the nodes v_i, v_{i+1}, ... , $v_{|P|}$.
        else
            Add w to P.
            Set w as *seen*.
            DFS(w).

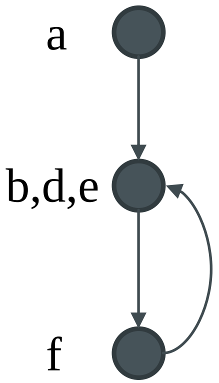
PATH-BASED DEPTH-FIRST SEARCH ALGORITHM

SCCs

{c}



P



Algorithm 1 Path-Based Depth-First Search Algorithm

Input: A directed graph $G = (V, L)$
Output: Set of SCCs represented by SCC (initially empty)
for $v \in V$ **do**
  Set the node as *unseen*.
Initiate a path $P = \{\}$
for $v \in V$ **do**
  if v is *unseen* **then**
    DFS(v).

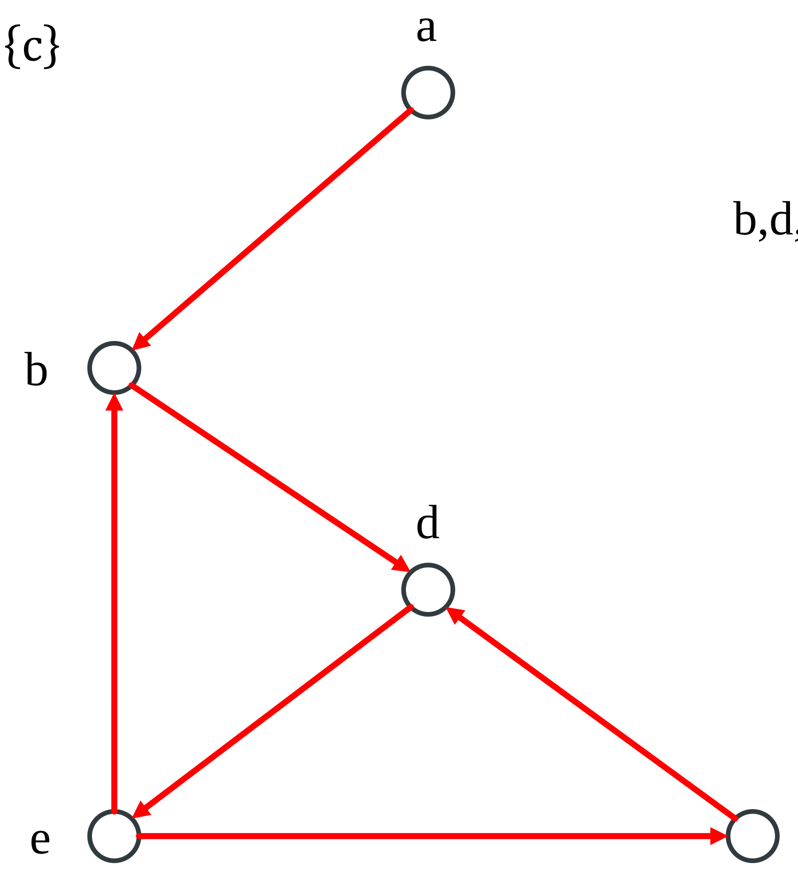
Algorithm 2 DFS(v)

if v has no outgoing arc **then**
  Consider v as an SCC and add it to SCC
  Set v as *seen*.
  Remove it from P.
else
  for arc $(v, w) \in E$ **do**
    if $w \in P$ $(v_i = w)$ **then**
      Contract the nodes v_i, v_{i+1}, ... , $v_{|P|}$.
    else
      Add w to P.
      Set w as *seen*.
      DFS(w).

PATH-BASED DEPTH-FIRST SEARCH ALGORITHM

SCCs

{c}



P

Algorithm 1 Path-Based Depth-First Search Algorithm

Input: A directed graph $G = (V, L)$
Output: Set of SCCs represented by SCC (initially empty)
for $v \in V$ **do**
  Set the node as *unseen*.
Initiate a path $P = \{\}$
for $v \in V$ **do**
  if v is *unseen* **then**
    DFS(v).

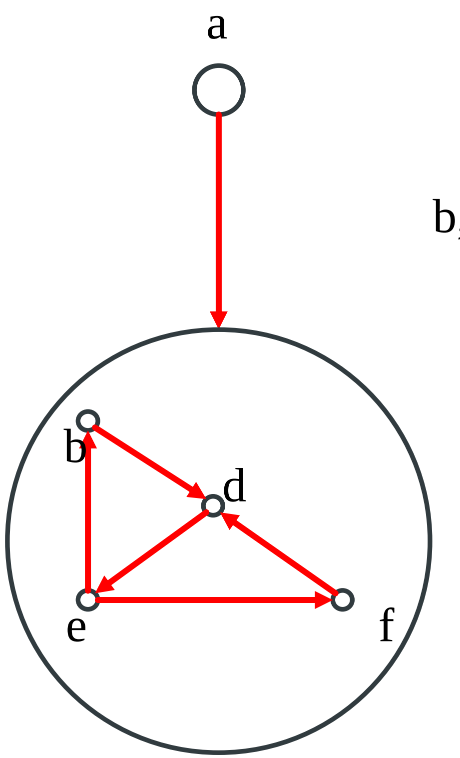
Algorithm 2 DFS(v)

if v has no outgoing arc **then**
  Consider v as an SCC and add it to SCC
  Set v as *seen*.
  Remove it from P.
else
  for arc $(v, w) \in E$ **do**
    if $w \in P$ $(v_i = w)$ **then**
      Contract the nodes v_i, v_{i+1}, ... , $v_{|P|}$.
    else
      Add w to P.
      Set w as *seen*.
      DFS(w).

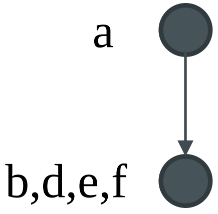
PATH-BASED DEPTH-FIRST SEARCH ALGORITHM

SCCs

{c}



P



Algorithm 1 Path-Based Depth-First Search Algorithm

Input: A directed graph $G = (V, L)$
Output: Set of SCCs represented by SCC (initially empty)
for $v \in V$ **do**
    Set the node as *unseen*.
Initiate a path $P = \{\}$
for $v \in V$ **do**
    if v is *unseen* **then**
        DFS(v).

Algorithm 2 DFS(v)

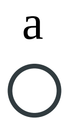
if v has no outgoing arc **then**
    Consider v as an SCC and add it to SCC
    Set v as *seen*.
    Remove it from P.
else
    for arc $(v, w) \in E$ **do**
        if $w \in P$ $(v_i = w)$ **then**
            Contract the nodes v_i, v_{i+1}, ... , $v_{|P|}$.
        else
            Add w to P.
            Set w as *seen*.
            DFS(w).

PATH-BASED DEPTH-FIRST SEARCH ALGORITHM

SCCs

{c}

{b,d,e,f}



P

Algorithm 1 Path-Based Depth-First Search Algorithm

Input: A directed graph $G = (V, L)$
Output: Set of SCCs represented by SCC (initially empty)
for $v \in V$ **do**
  Set the node as *unseen*.
Initiate a path $P = \{\}$
for $v \in V$ **do**
  if v is *unseen* **then**
    DFS(v).

Algorithm 2 DFS(v)

if v has no outgoing arc **then**
  Consider v as an SCC and add it to SCC
  Set v as *seen*.
  Remove it from P.
else
  for arc $(v, w) \in E$ **do**
    if $w \in P$ $(v_i = w)$ **then**
      Contract the nodes v_i, v_{i+1}, ... , $v_{|P|}$.
    else
      Add w to P.
      Set w as *seen*.
      DFS(w).

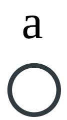
PATH-BASED DEPTH-FIRST SEARCH ALGORITHM

SCCs

{c}

{b,d,e,f}

{a}



P

Algorithm 1 Path-Based Depth-First Search Algorithm

Input: A directed graph $G = (V, L)$
Output: Set of SCCs represented by SCC (initially empty)
for $v \in V$ **do**
    Set the node as *unseen*.
Initiate a path $P = \{\}$
for $v \in V$ **do**
    if v is *unseen* **then**
        DFS(v).

Algorithm 2 DFS(v)

if v has no outgoing arc **then**
    Consider v as an SCC and add it to SCC
    Set v as *seen*.
    Remove it from P.
else
    for arc $(v, w) \in E$ **do**
        if $w \in P$ $(v_i = w)$ **then**
            Contract the nodes v_i, v_{i+1}, ... , $v_{|P|}$.
        else
            Add w to P.
            Set w as *seen*.
            DFS(w).

PATH-BASED DEPTH-FIRST SEARCH ALGORITHM

SCCs **P**

{c}

{b,d,e,f}

{a}

Algorithm 1 Path-Based Depth-First Search Algorithm

Input: A directed graph $G = (V, L)$
Output: Set of SCCs represented by SCC (initially empty)
for $v \in V$ **do**
    Set the node as *unseen*.
Initiate a path $P = \{\}$
for $v \in V$ **do**
    if v is *unseen* **then**
        DFS(v).

Algorithm 2 DFS(v)

if v has no outgoing arc **then**
    Consider v as an SCC and add it to SCC
    Set v as *seen*.
    Remove it from P.
else
    for arc $(v, w) \in E$ **do**
        if $w \in P$ $(v_i = w)$ **then**
            Contract the nodes v_i, v_{i+1}, ... , $v_{|P|}$.
        else
            Add w to P.
            Set w as *seen*.
            DFS(w).

Complexity

$$O(E + V)$$

Algorithm 1 Path-Based Depth-First Search Algorithm

- **Input:** A directed graph $G = (V, L)$
- **Output:** Set of SCCs represented by SCC (initially empty)
- **for** $v \in V$ **do**
  - Set the node as *unseen*.
- Initiate a path $P = \{\}$
- **for** $v \in V$ **do**
  - **if** v is *unseen* **then**
    - DFS(v).

Algorithm 2 DFS(v)

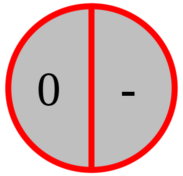
- **if** v has no outgoing arc **then**
  - Consider v as an SCC and add it to SCC
  - Set v as *seen*.
  - Remove it from P.
- **else**
  - **for** arc $(v, w) \in E$ **do**
    - **if** $w \in P$ $(v_i = w)$ **then**
      - Contract the nodes v_i, v_{i+1}, ... , $v_{|P|}$.
    - **else**
      - Add w to P.
      - Set w as *seen*.
      - DFS(w).

KOSARAJU-SHARIR ALGORITHM

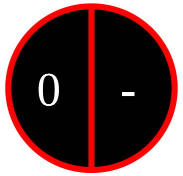
Algorithm 3 Kosaraju-Sharir Algorithm

Input: A directed graph $G = (V, L)$
Output: Set of SCCs represented by SCC (initially empty)
Run DFS on Graph G to compute finishing time (reverse postorder) of each node.
$V^R = V$, $L^R = \emptyset$.
Build $G^R = (V^R, L^R)$
for $l = (u, v) \in L$ **do**
    $L^R = L^R \cup (v, u)$
Run DFS on G^R, considering vertices in the decreasing reverse postorder.
Output the vertices of each Depth First Traversal as an SCC.

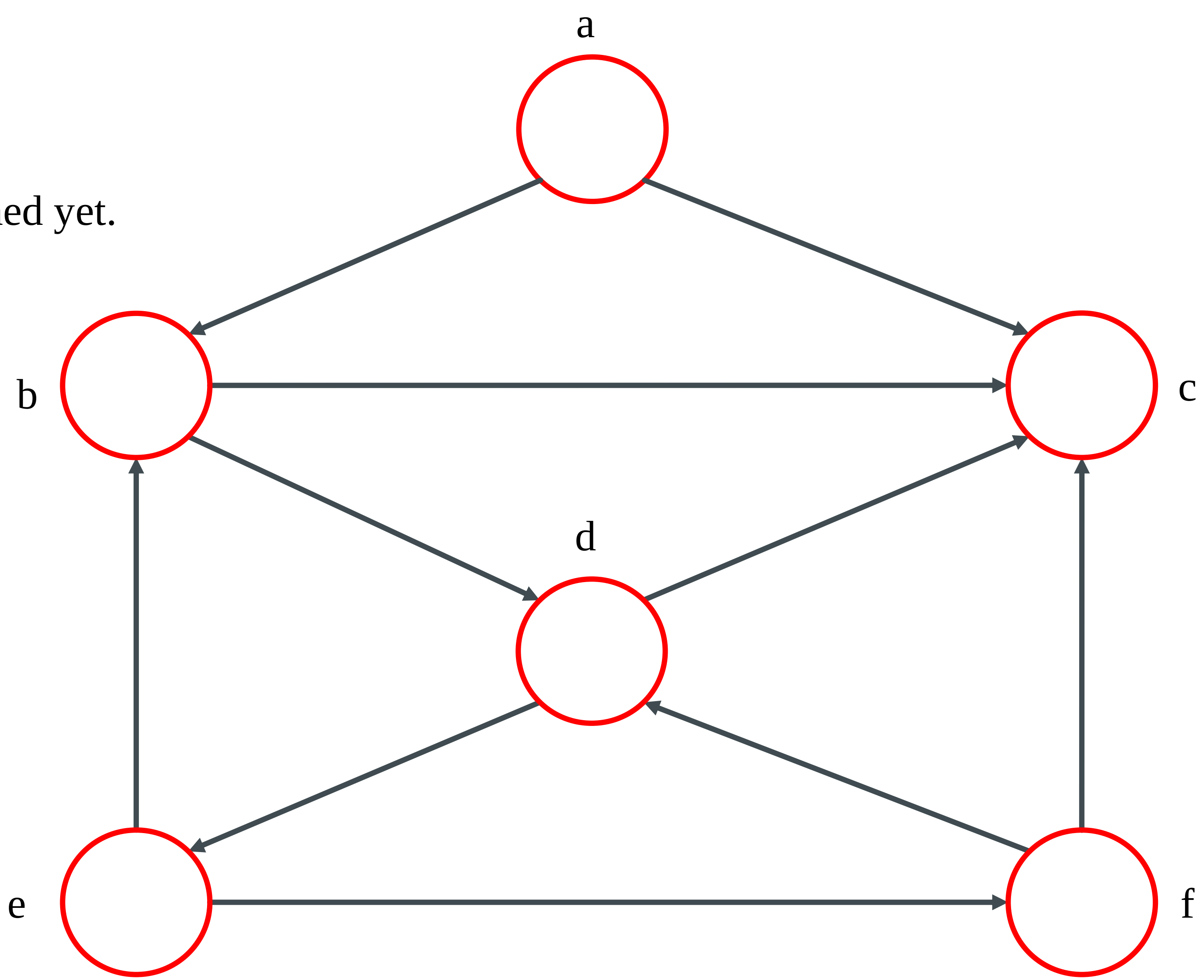
KOSARAJU-SHARIR ALGORITHM



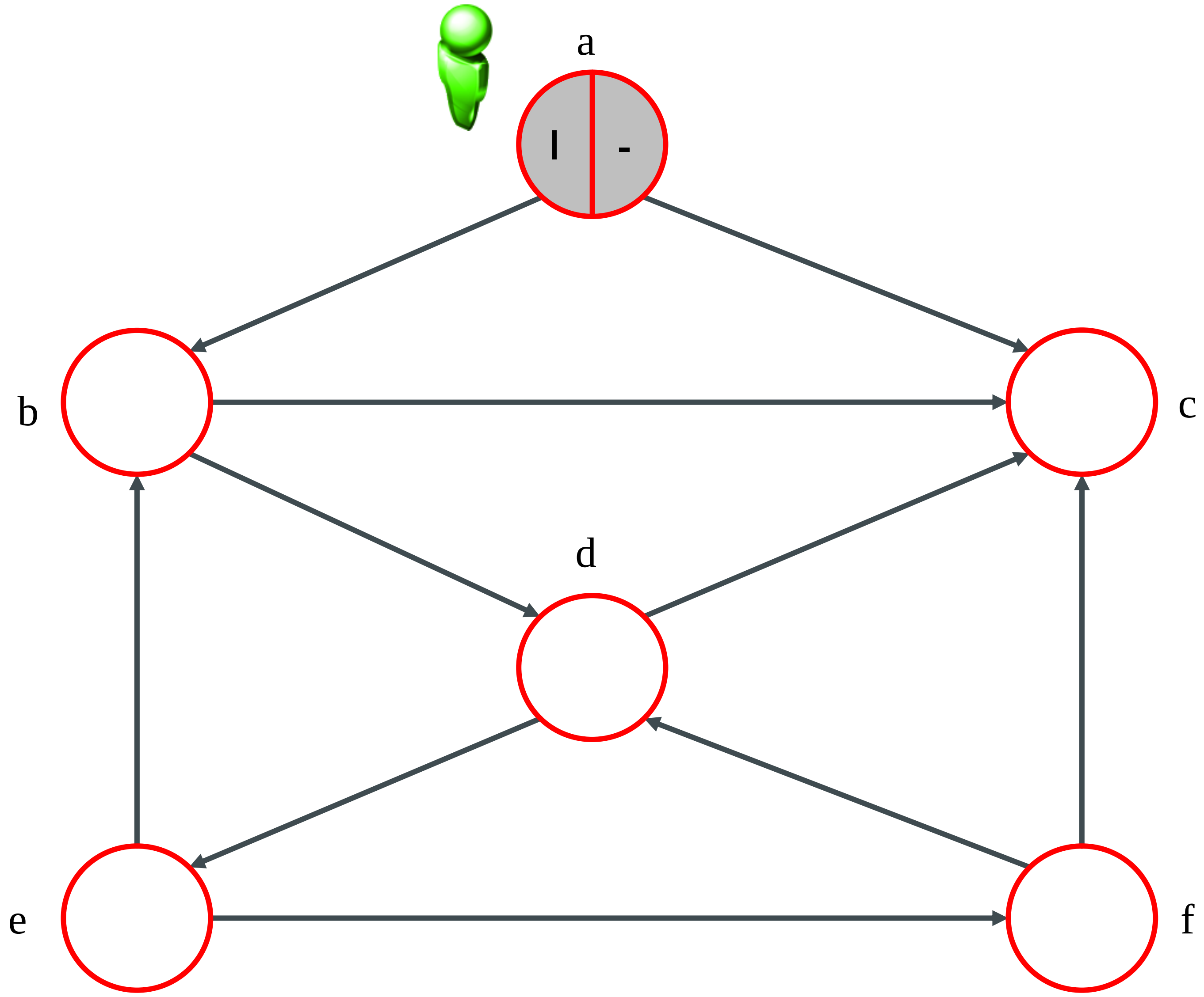
Seen, not finished yet.



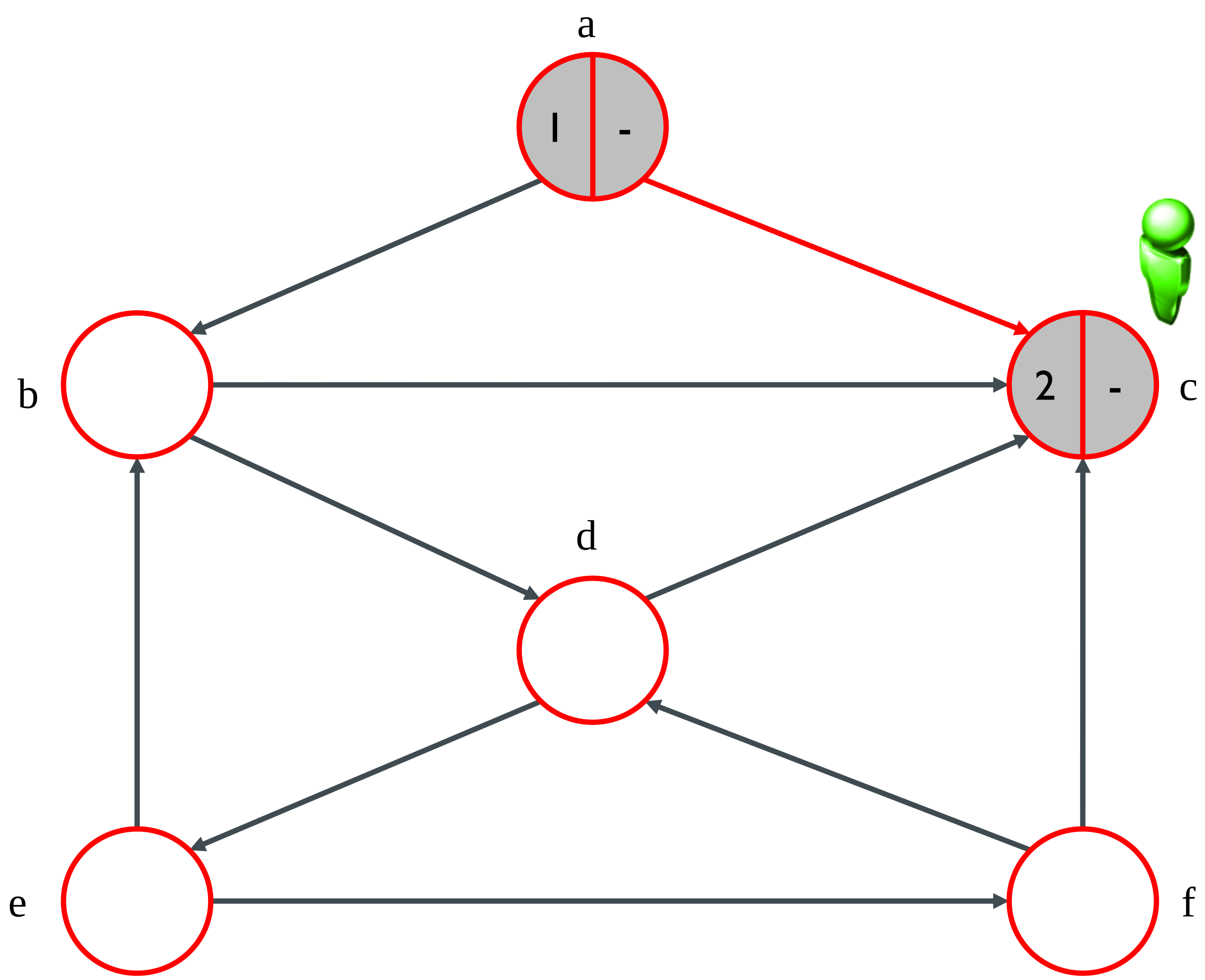
Finished.



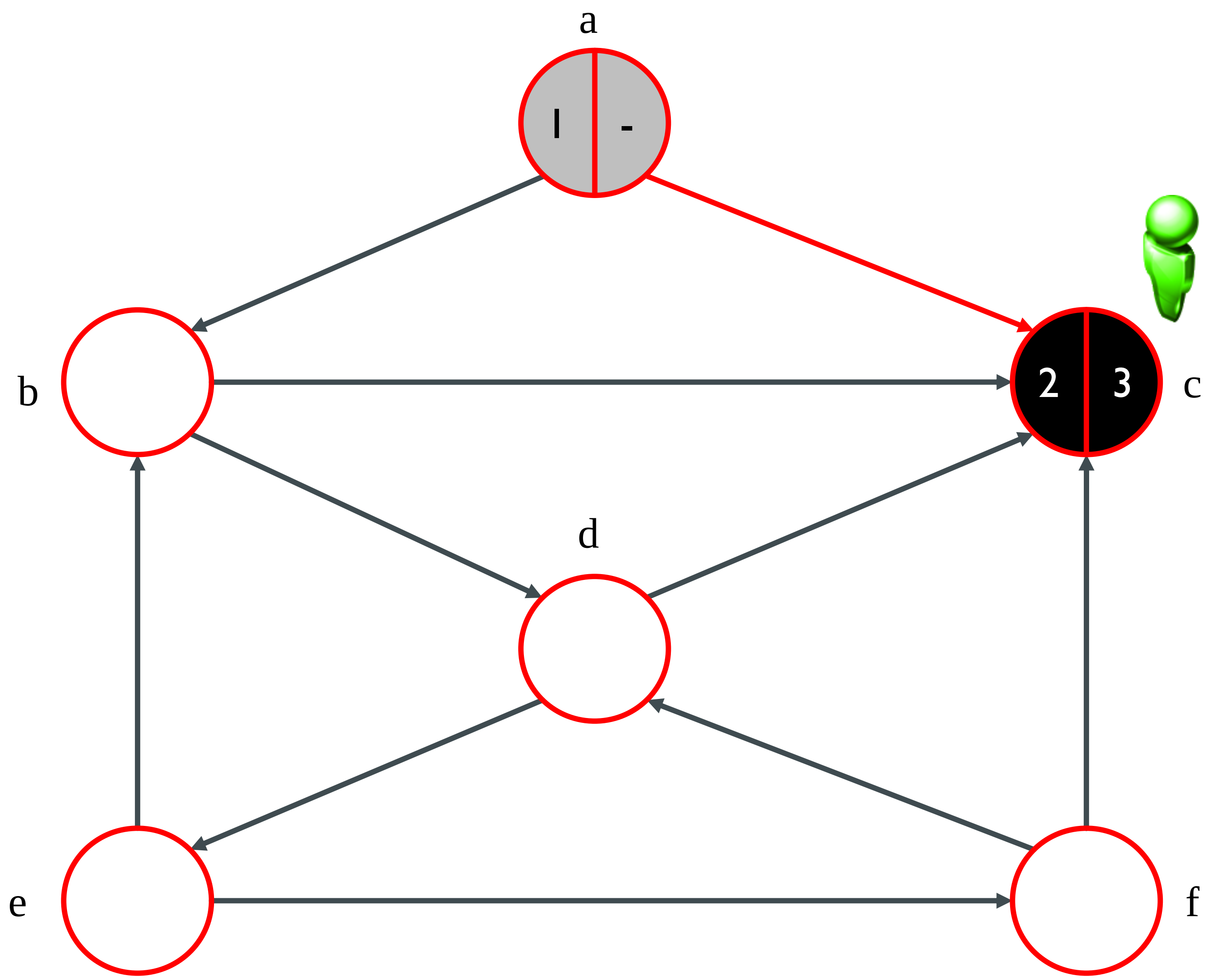
KOSARAJU-SHARIR ALGORITHM



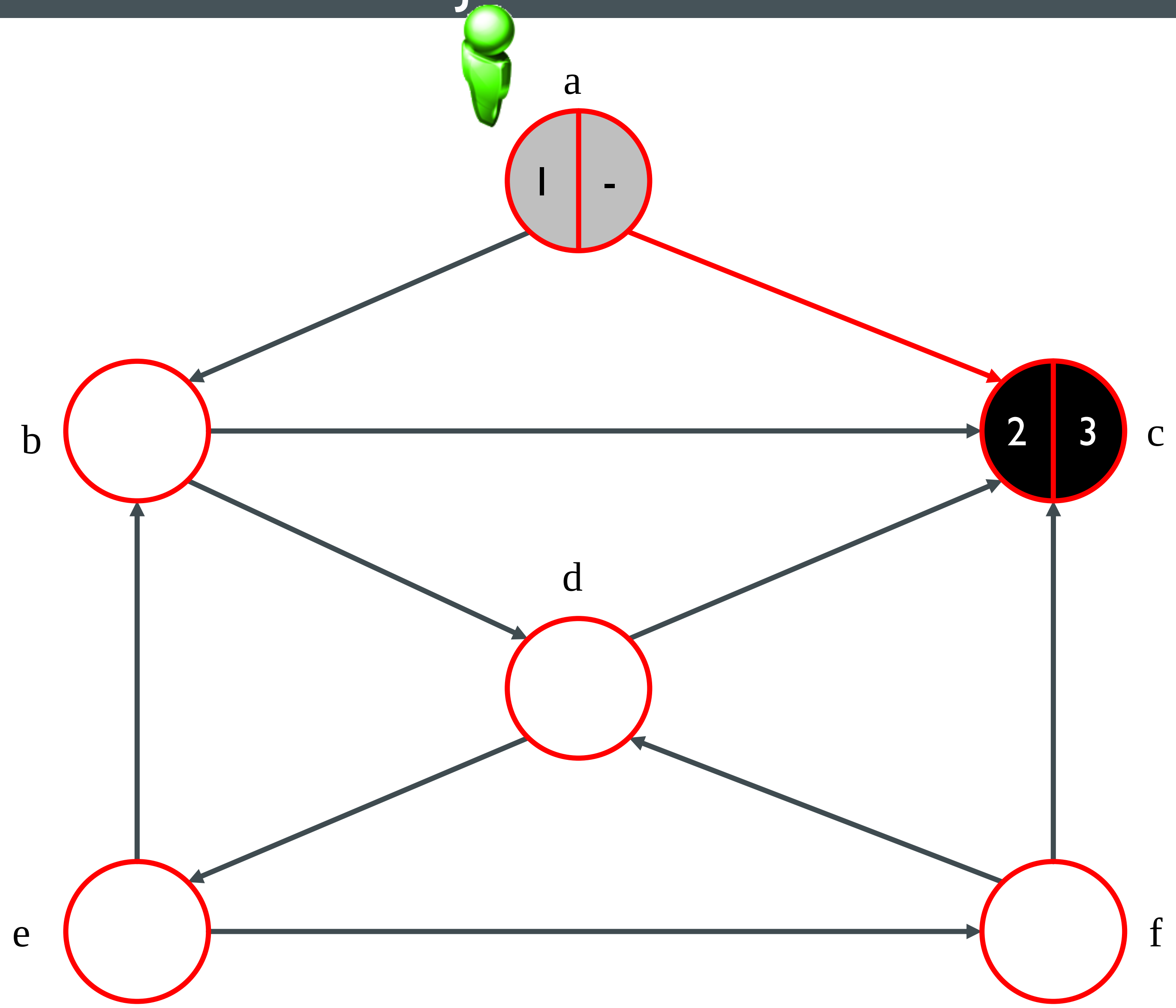
KOSARAJU-SHARIR ALGORITHM



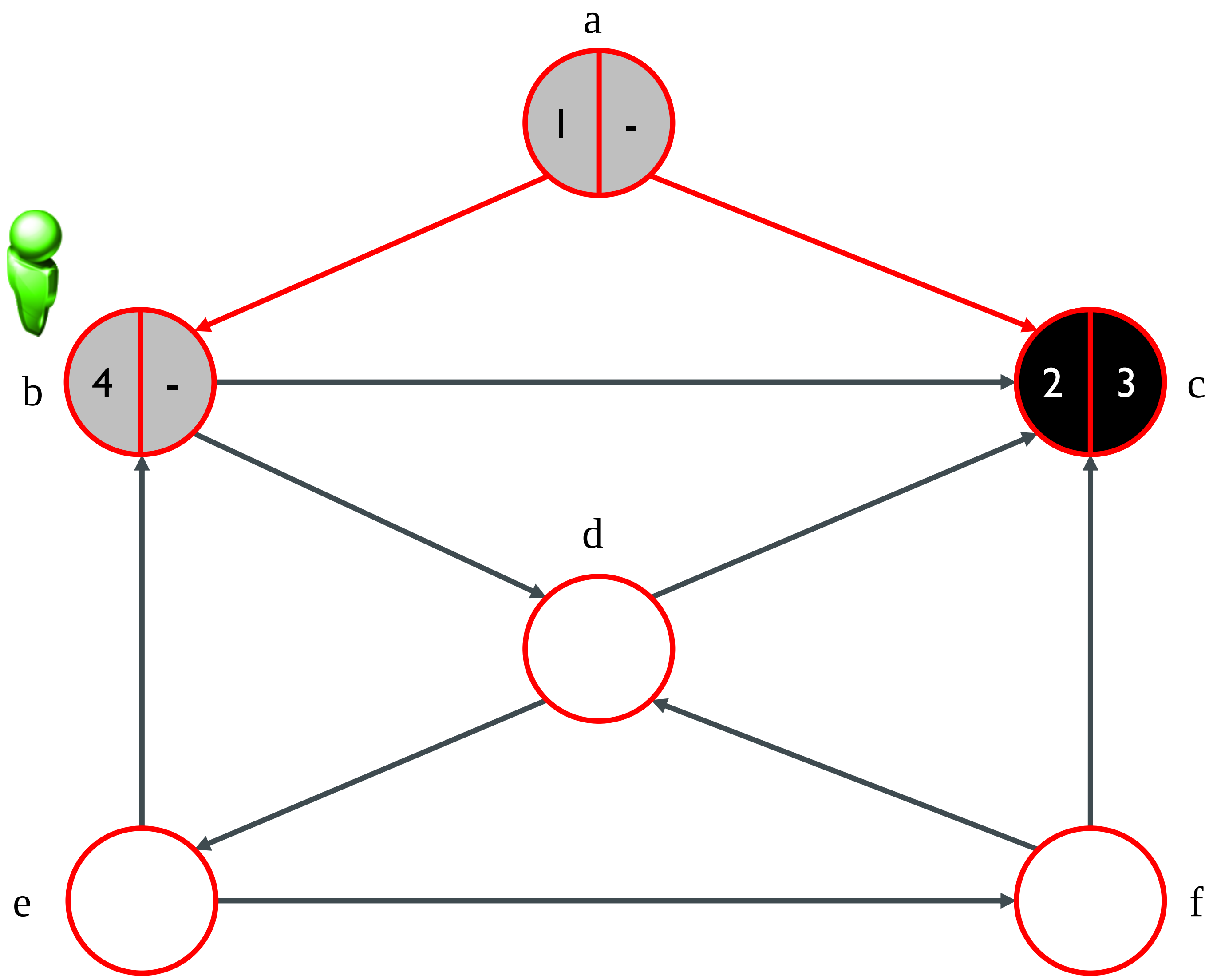
KOSARAJU-SHARIR ALGORITHM



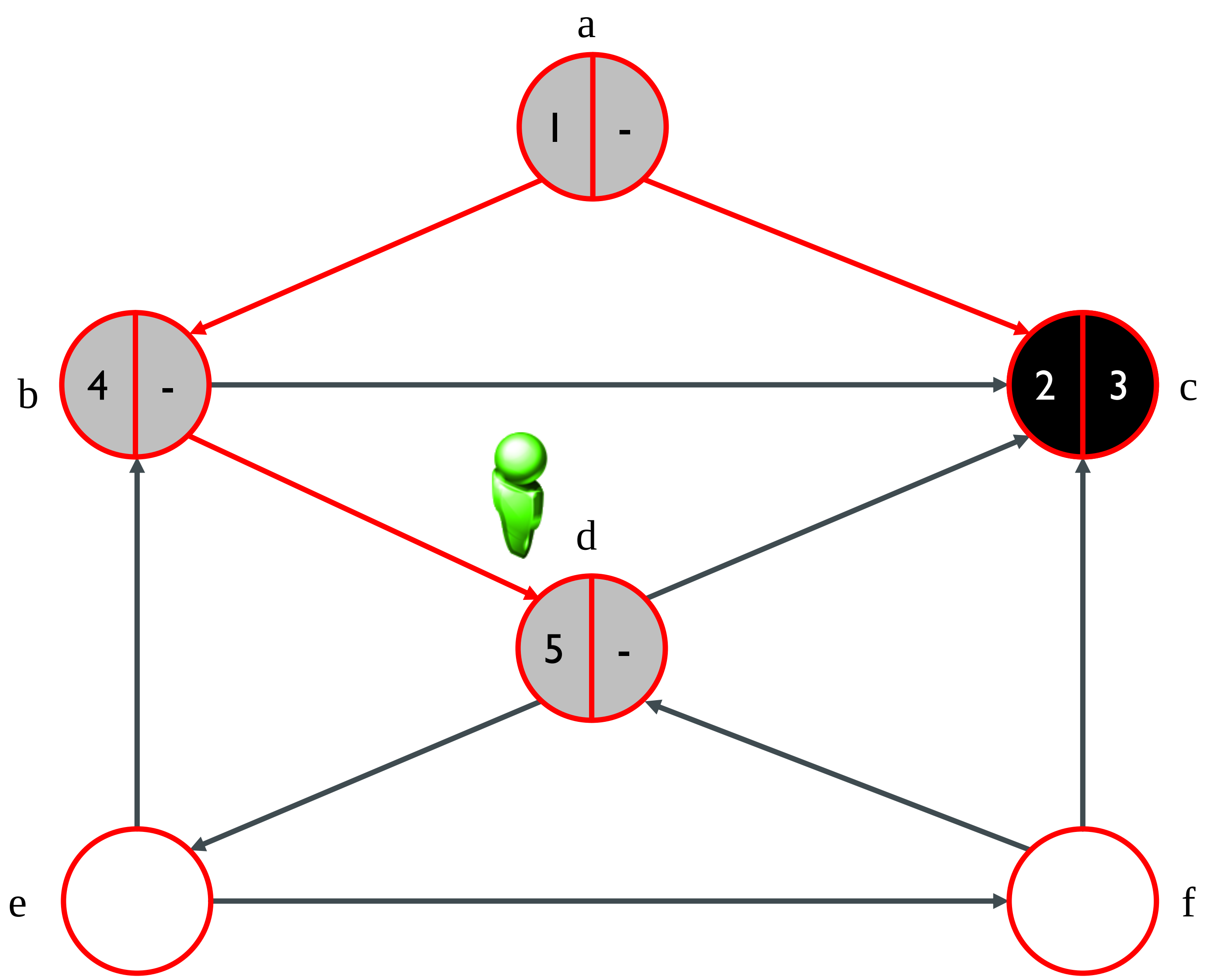
KOSARAJU-SHARIR ALGORITHM



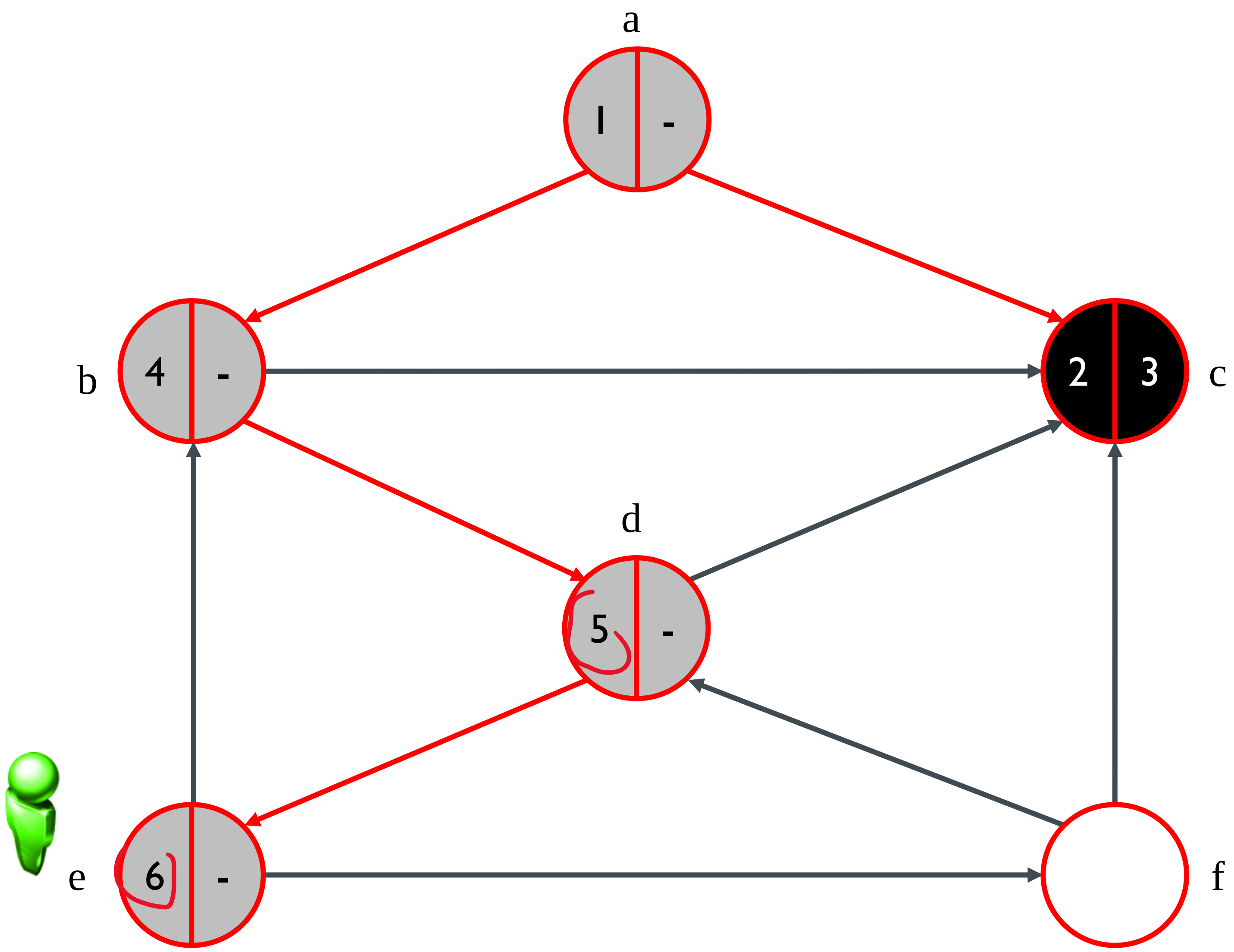
KOSARAJU-SHARIR ALGORITHM



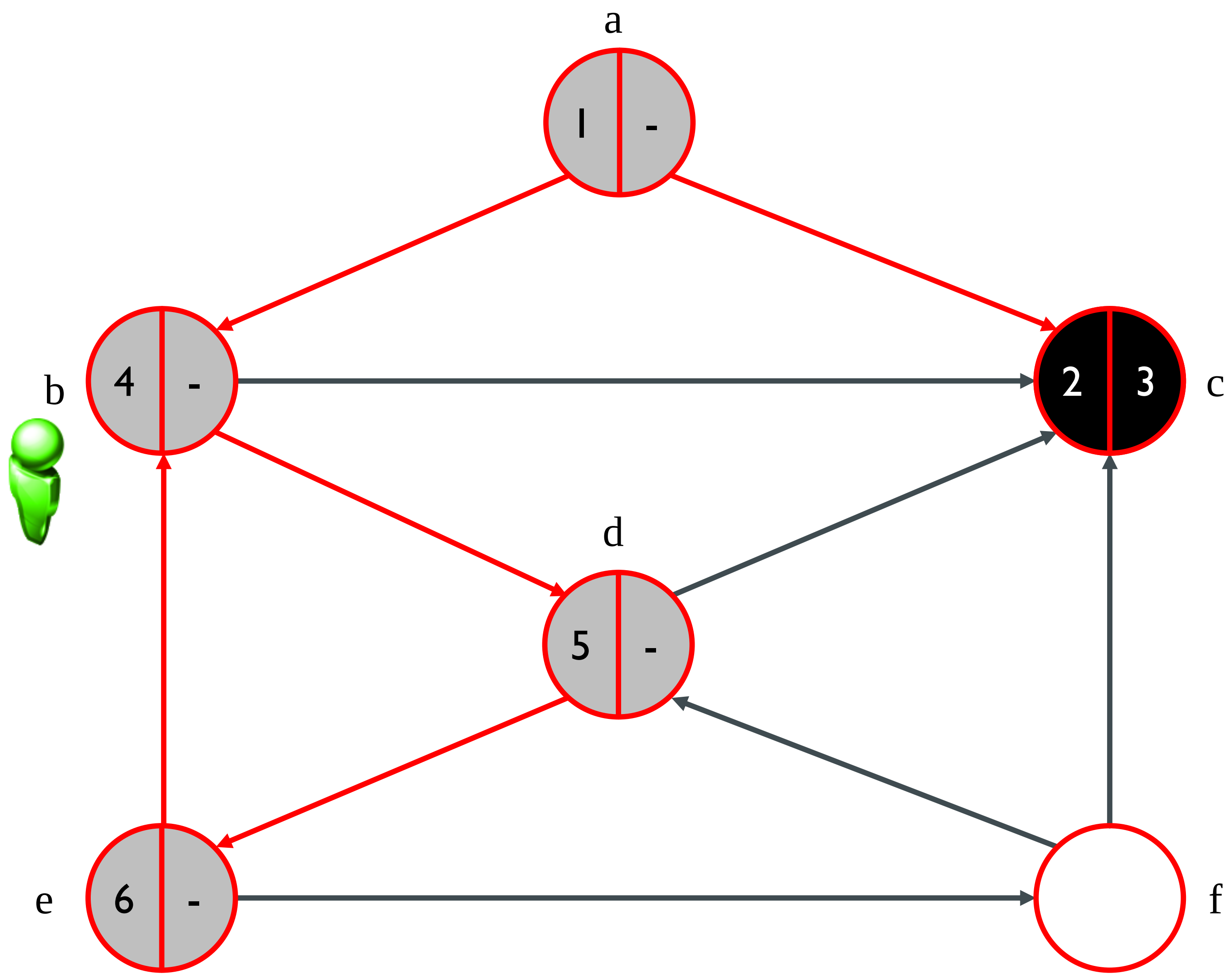
KOSARAJU-SHARIR ALGORITHM



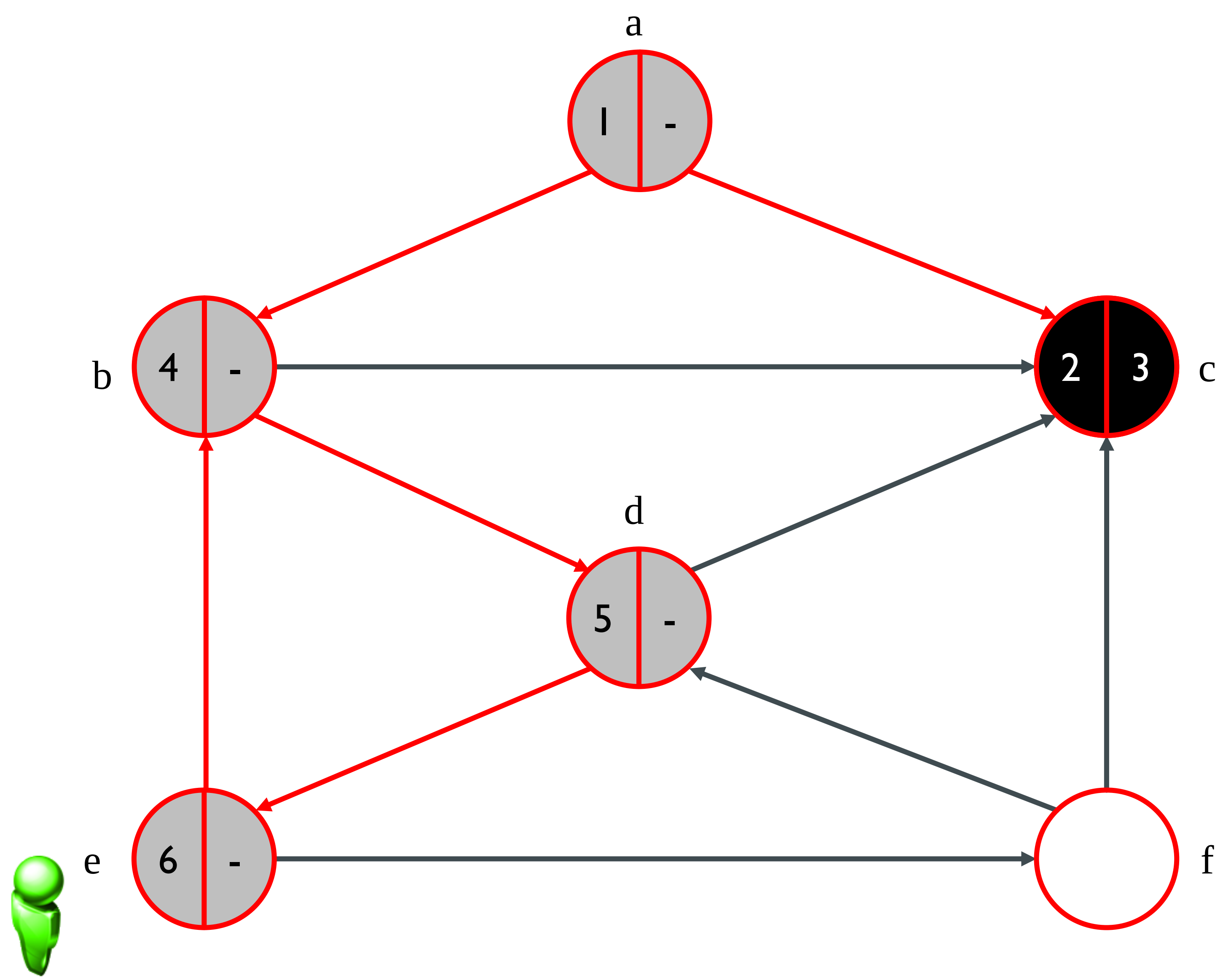
KOSARAJU-SHARIR ALGORITHM



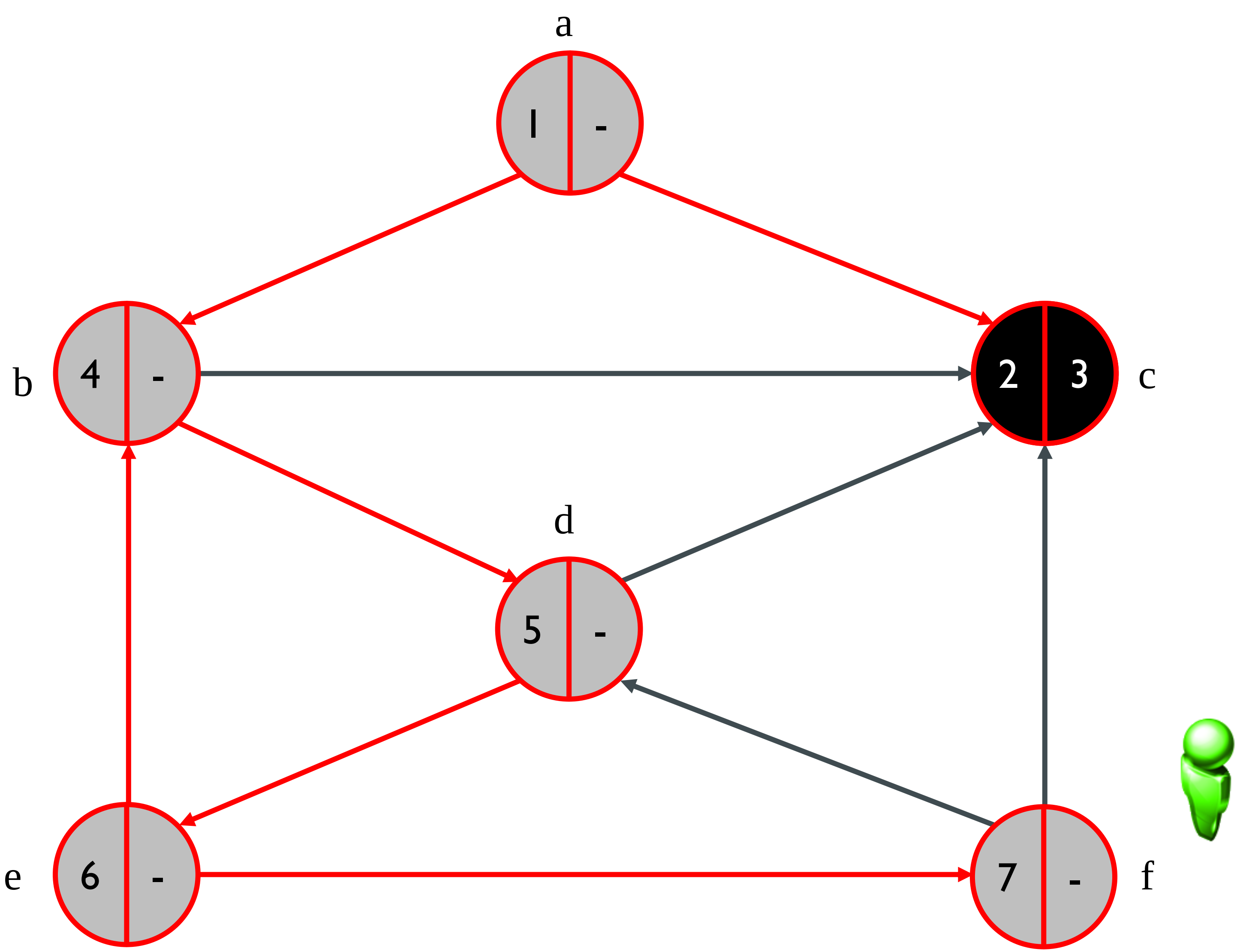
KOSARAJU-SHARIR ALGORITHM



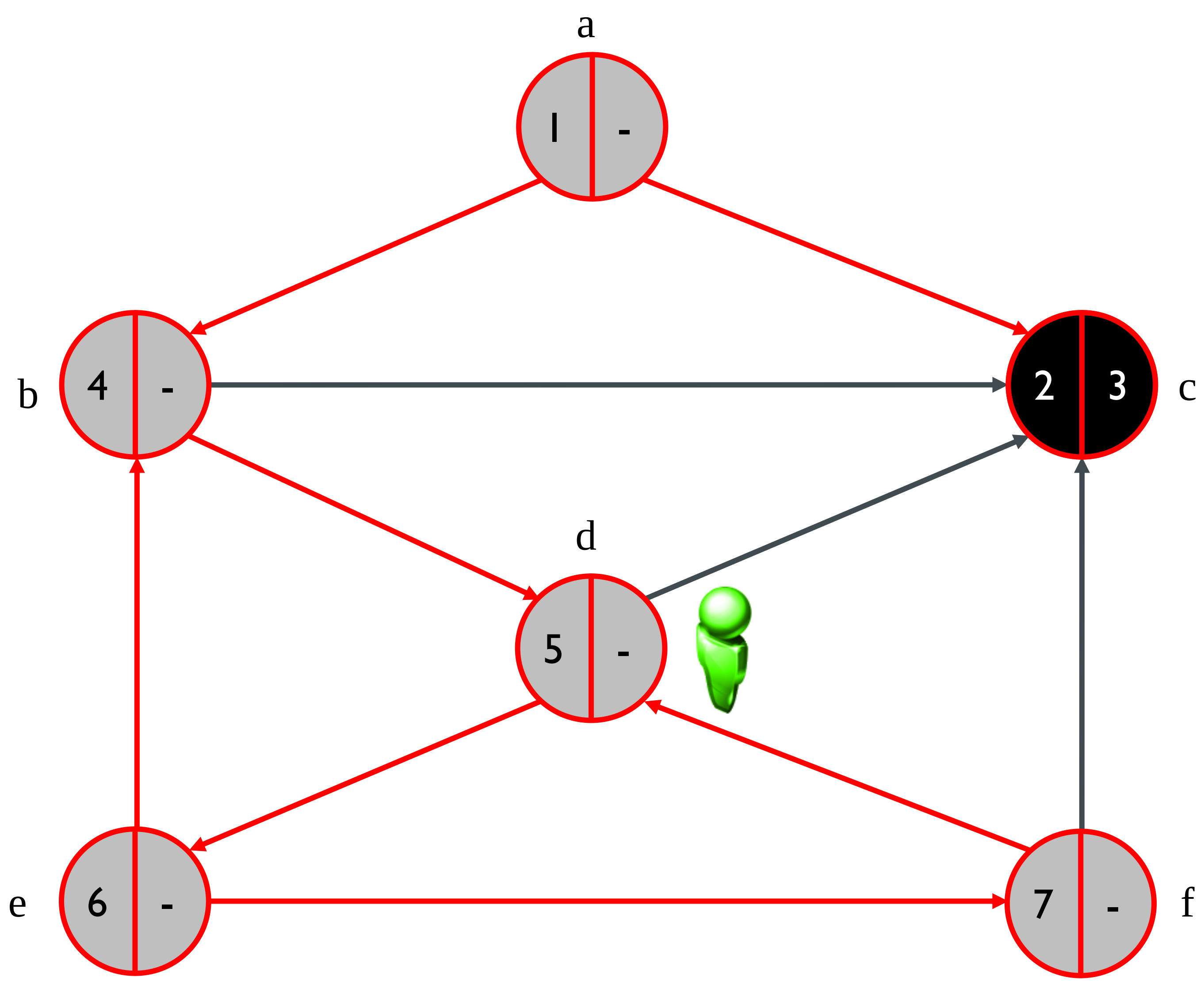
KOSARAJU-SHARIR ALGORITHM



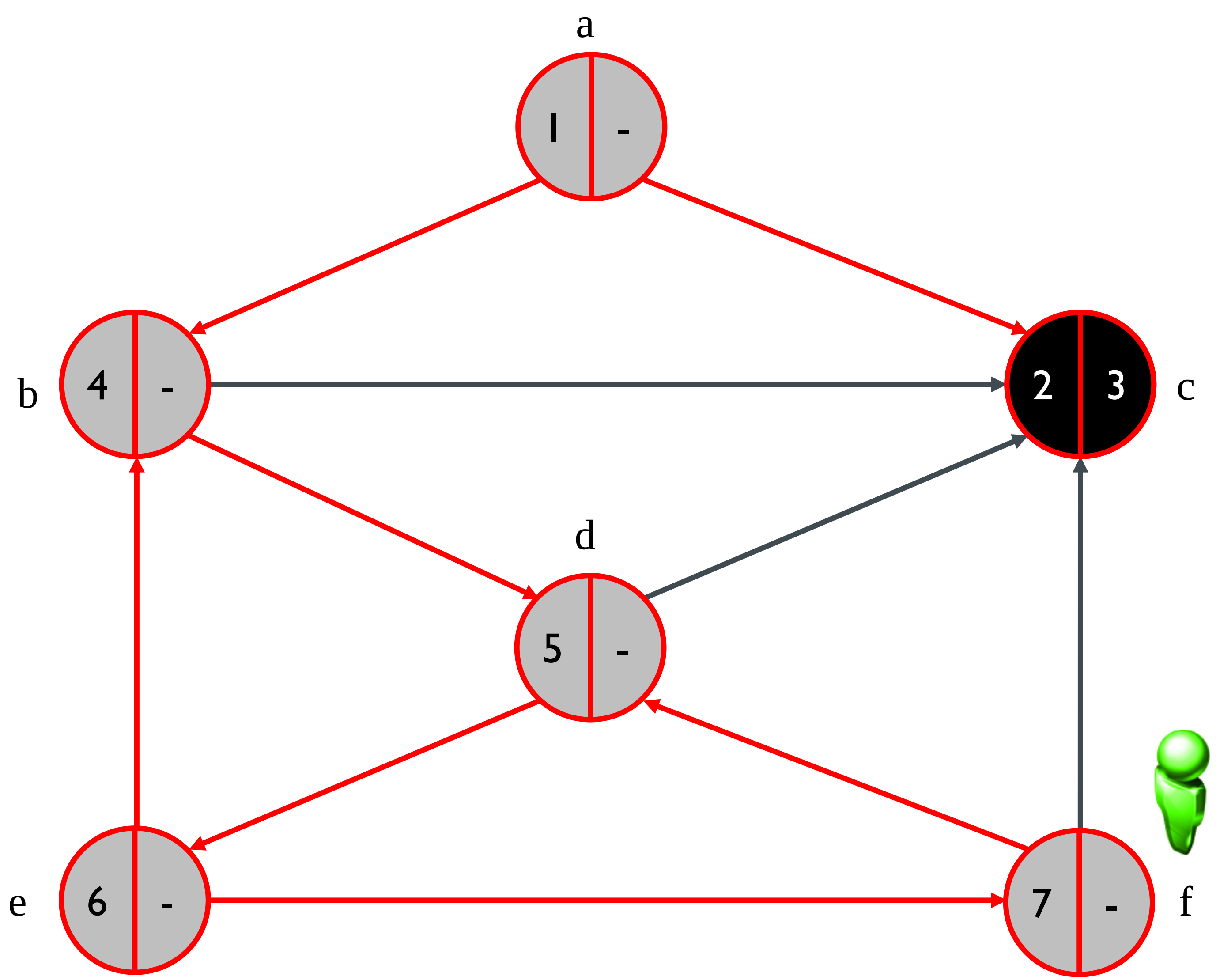
KOSARAJU-SHARIR ALGORITHM



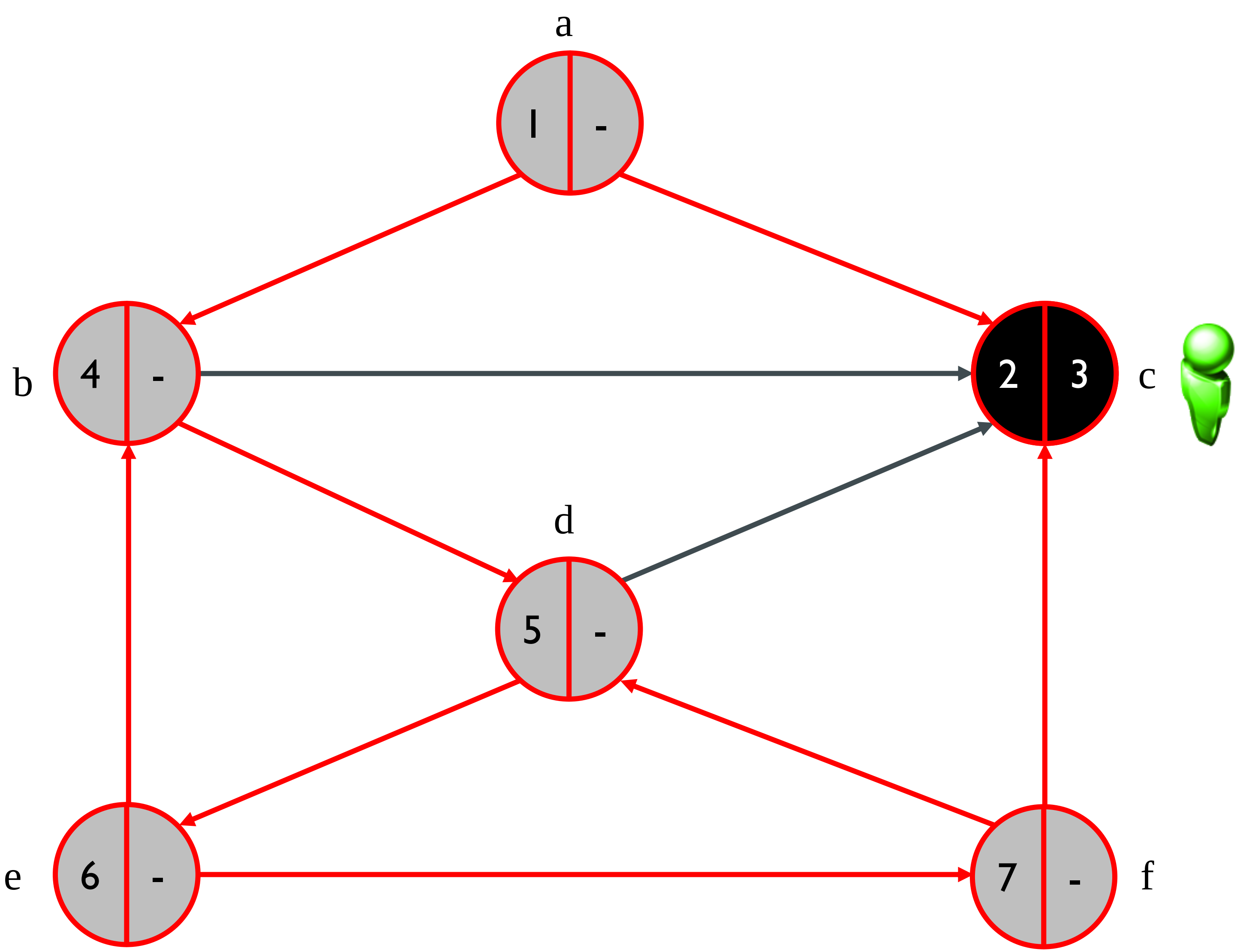
KOSARAJU-SHARIR ALGORITHM



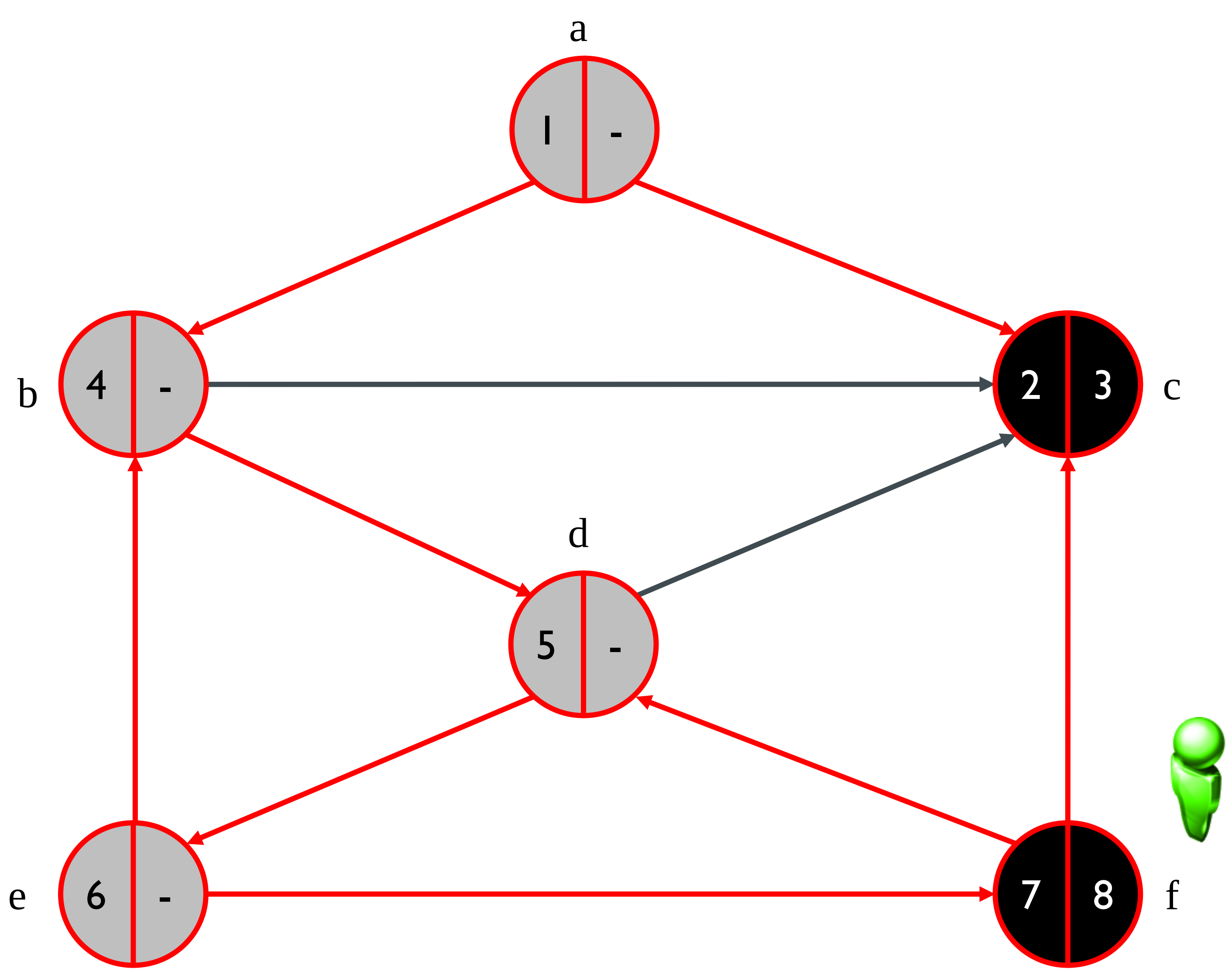
KOSARAJU-SHARIR ALGORITHM



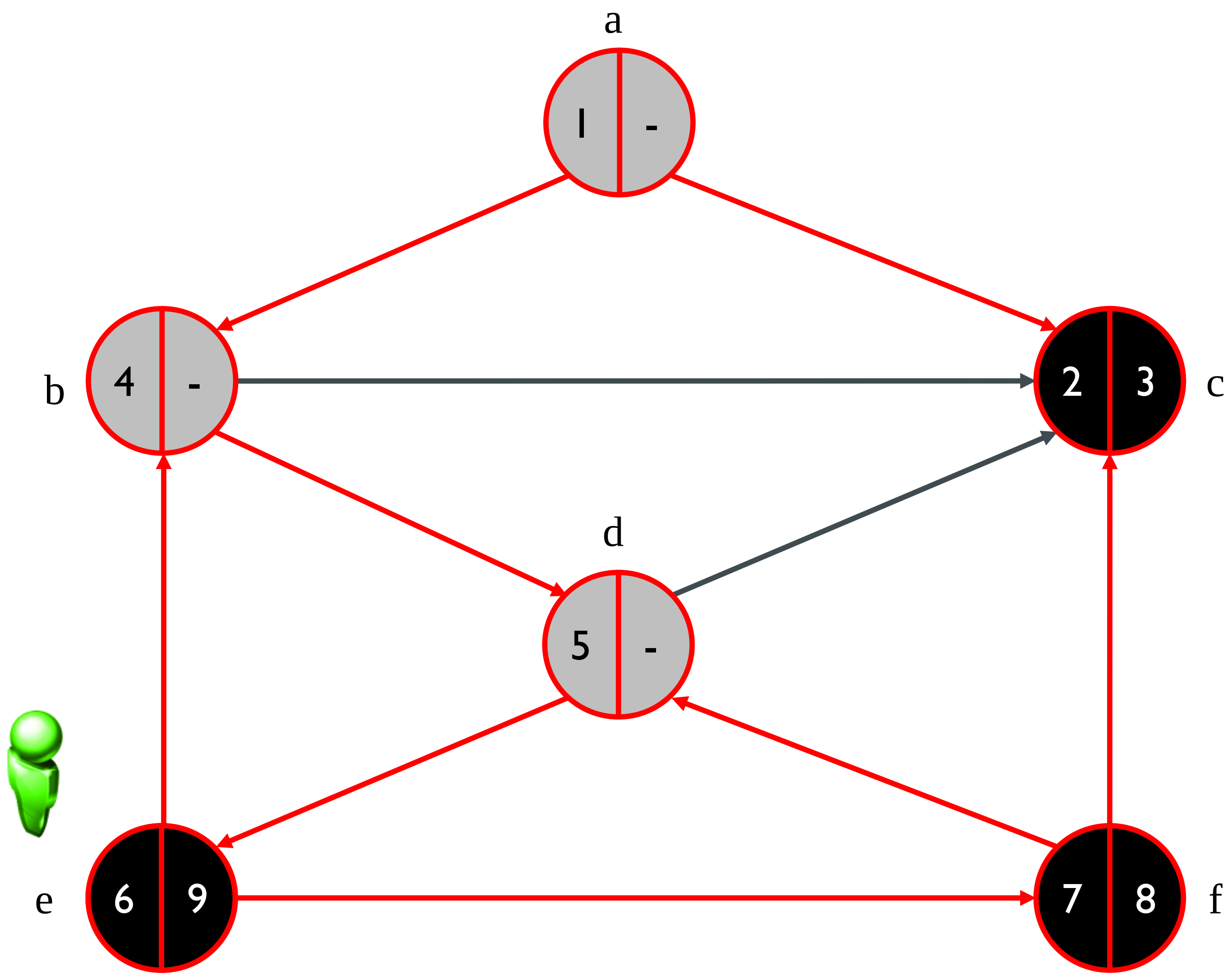
KOSARAJU-SHARIR ALGORITHM



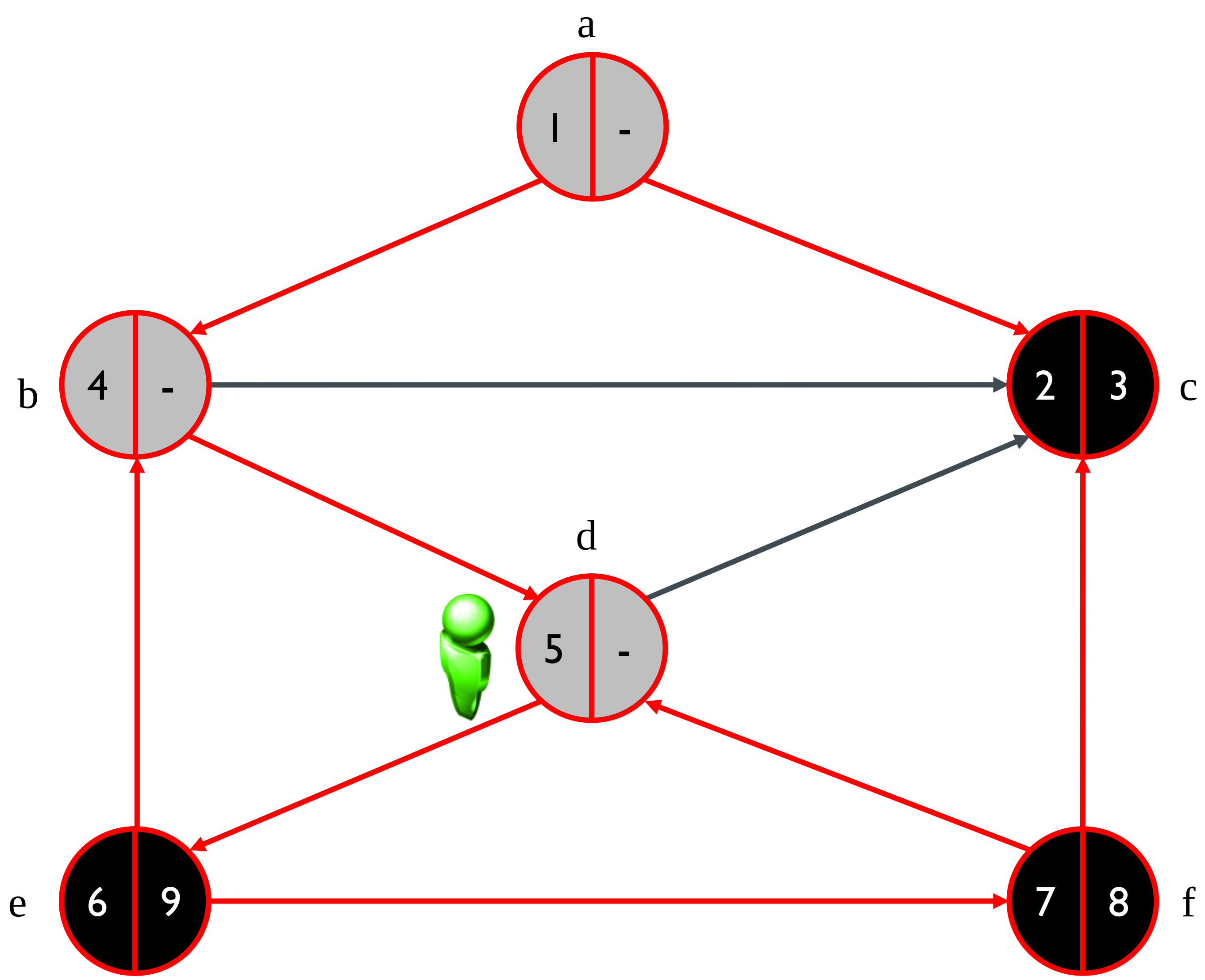
KOSARAJU-SHARIR ALGORITHM



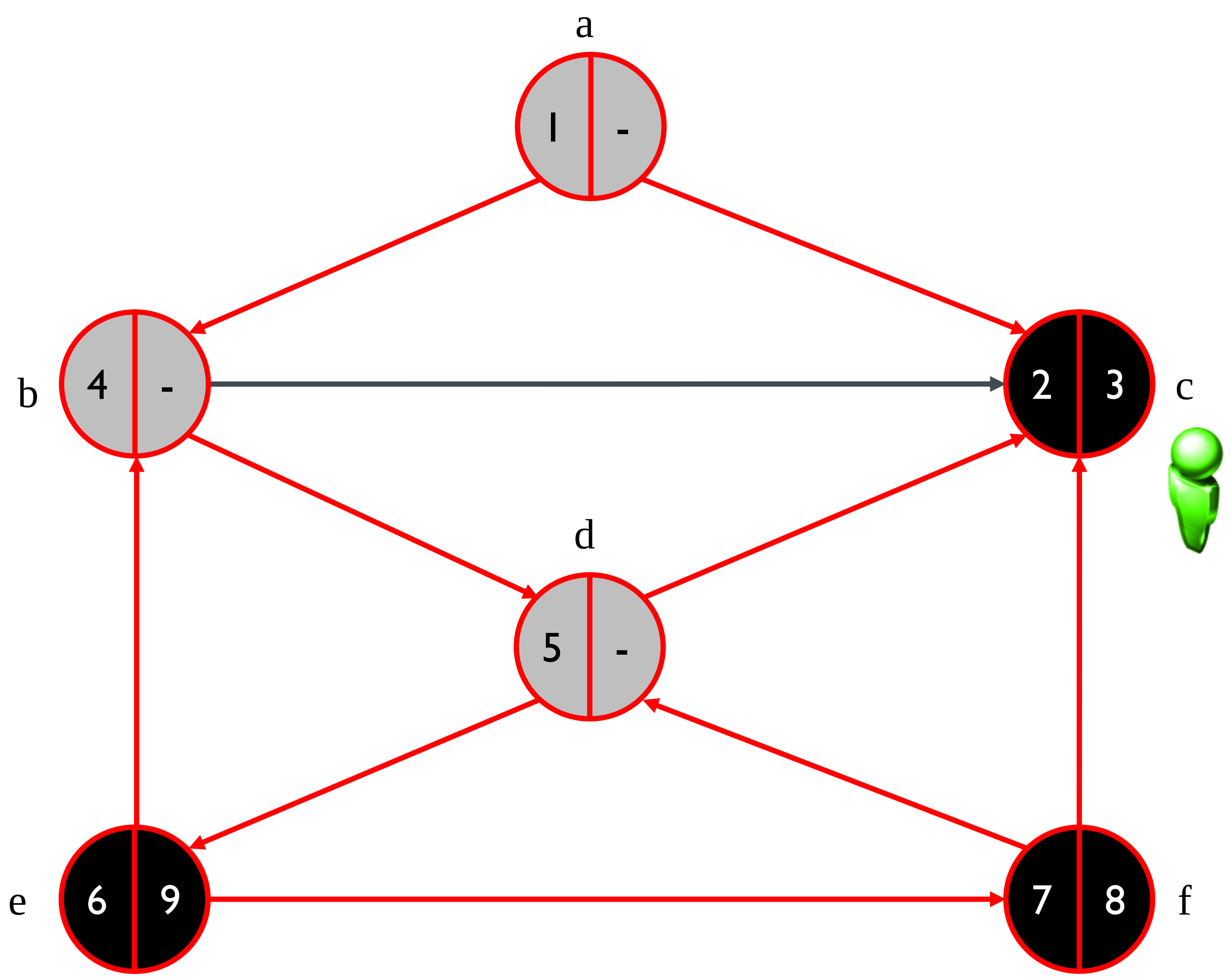
KOSARAJU-SHARIR ALGORITHM



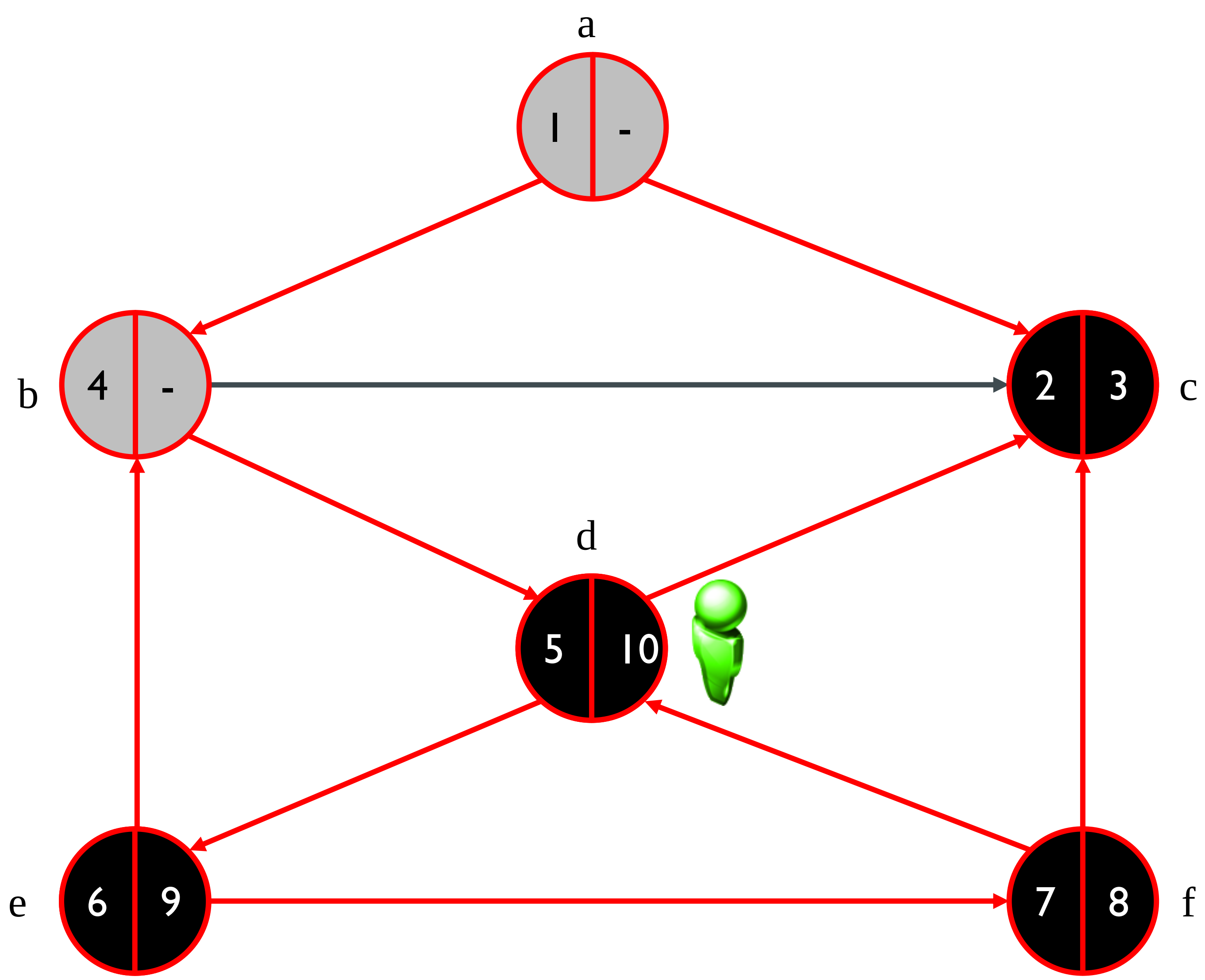
KOSARAJU-SHARIR ALGORITHM



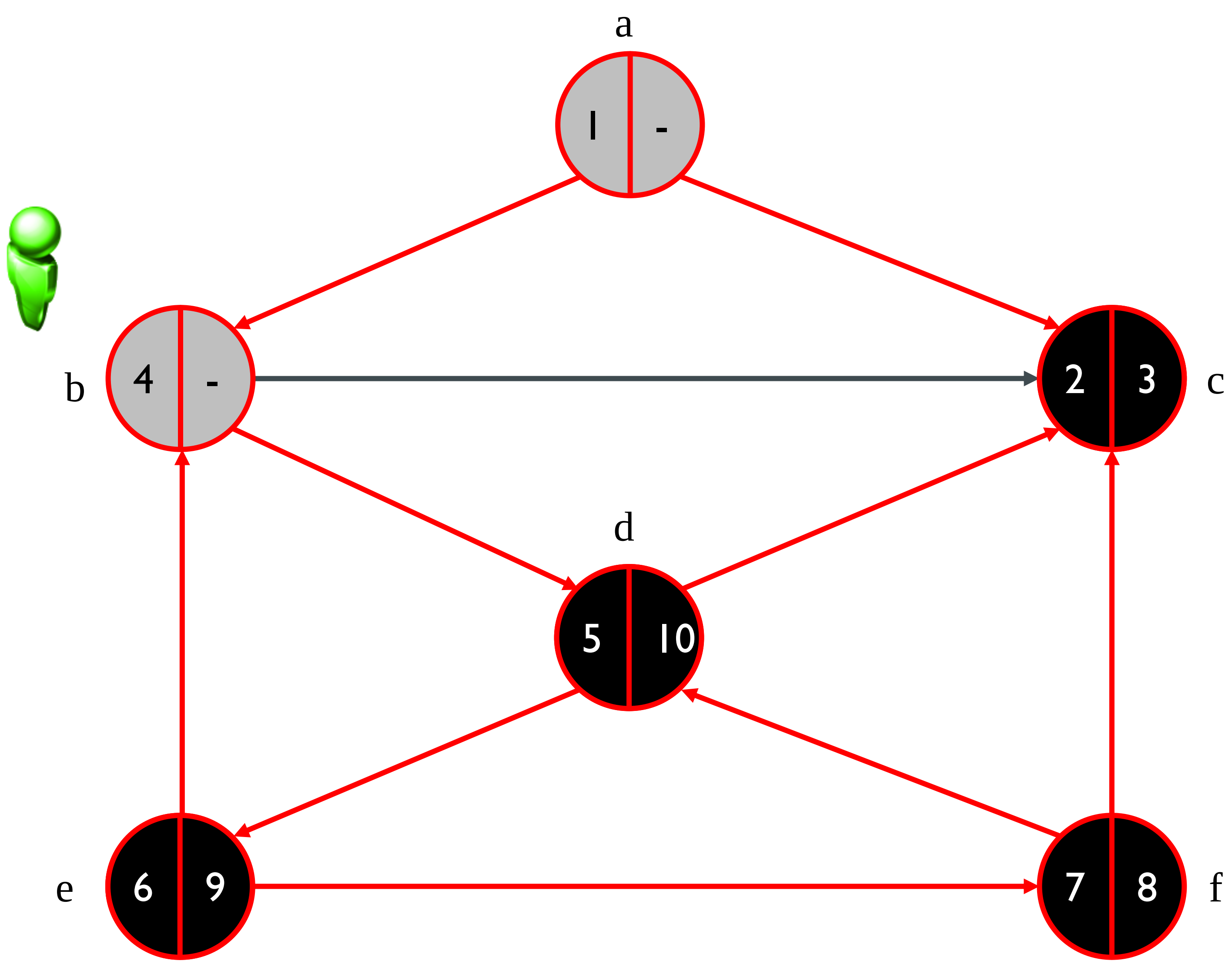
KOSARAJU-SHARIR ALGORITHM



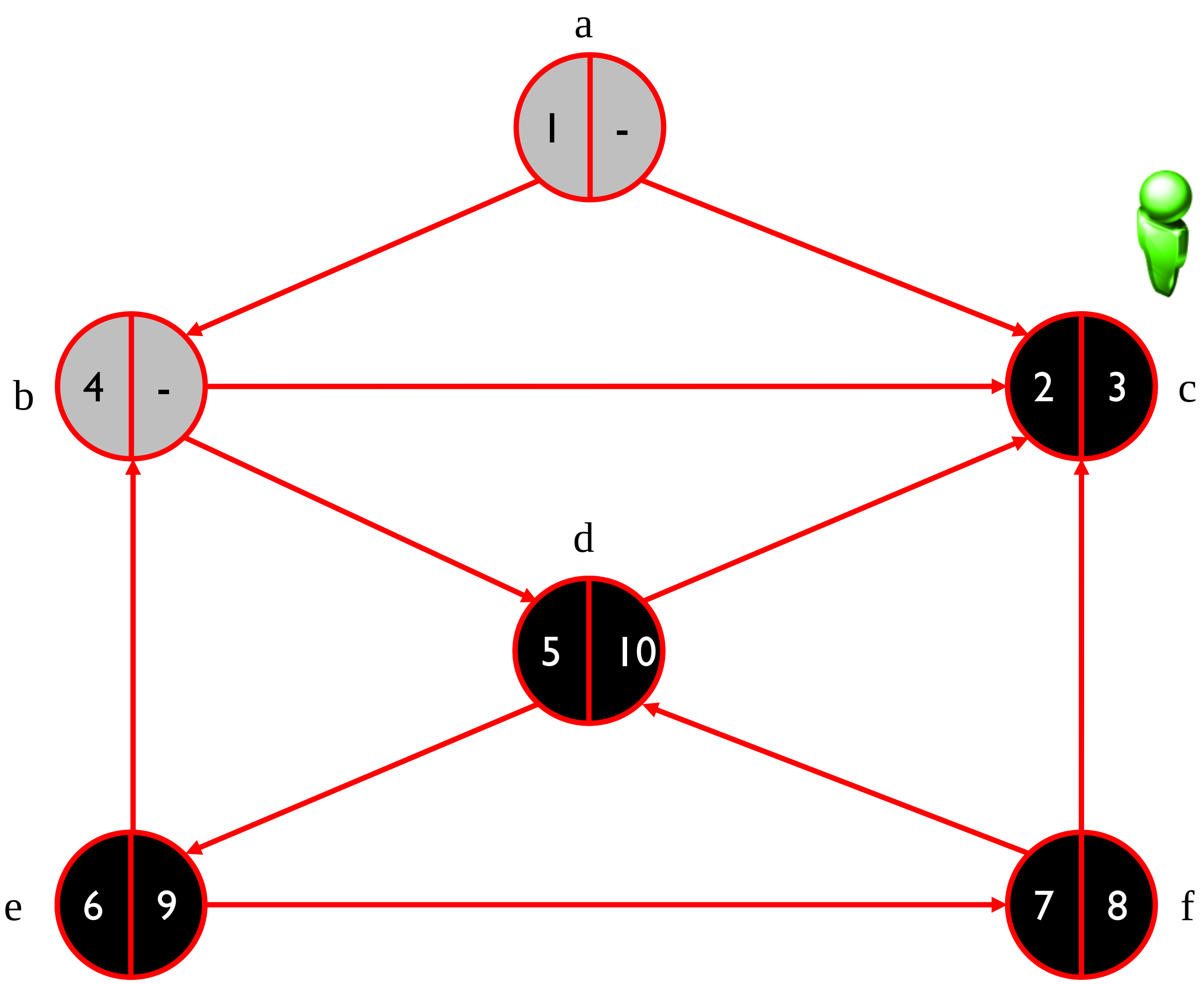
KOSARAJU-SHARIR ALGORITHM



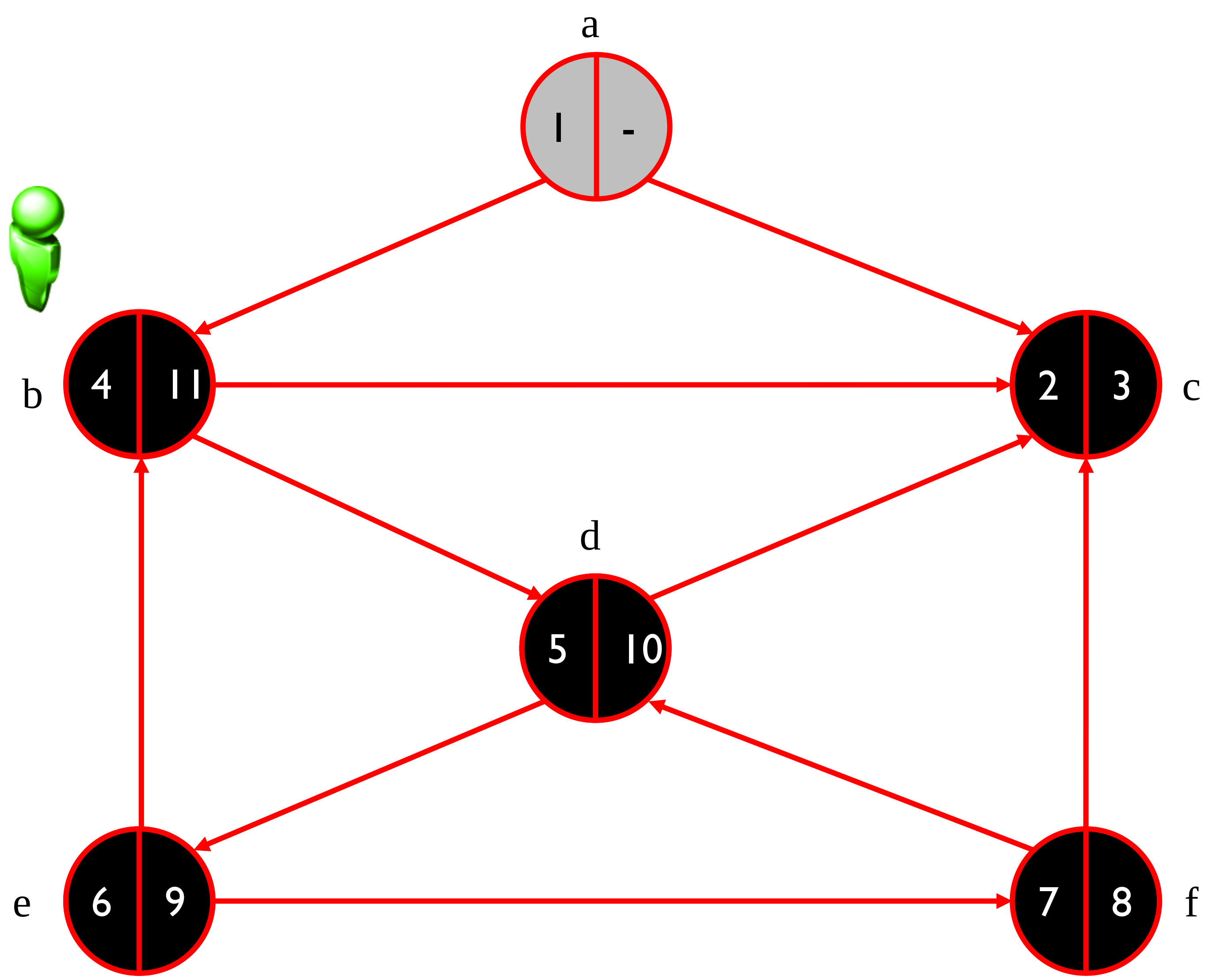
KOSARAJU-SHARIR ALGORITHM



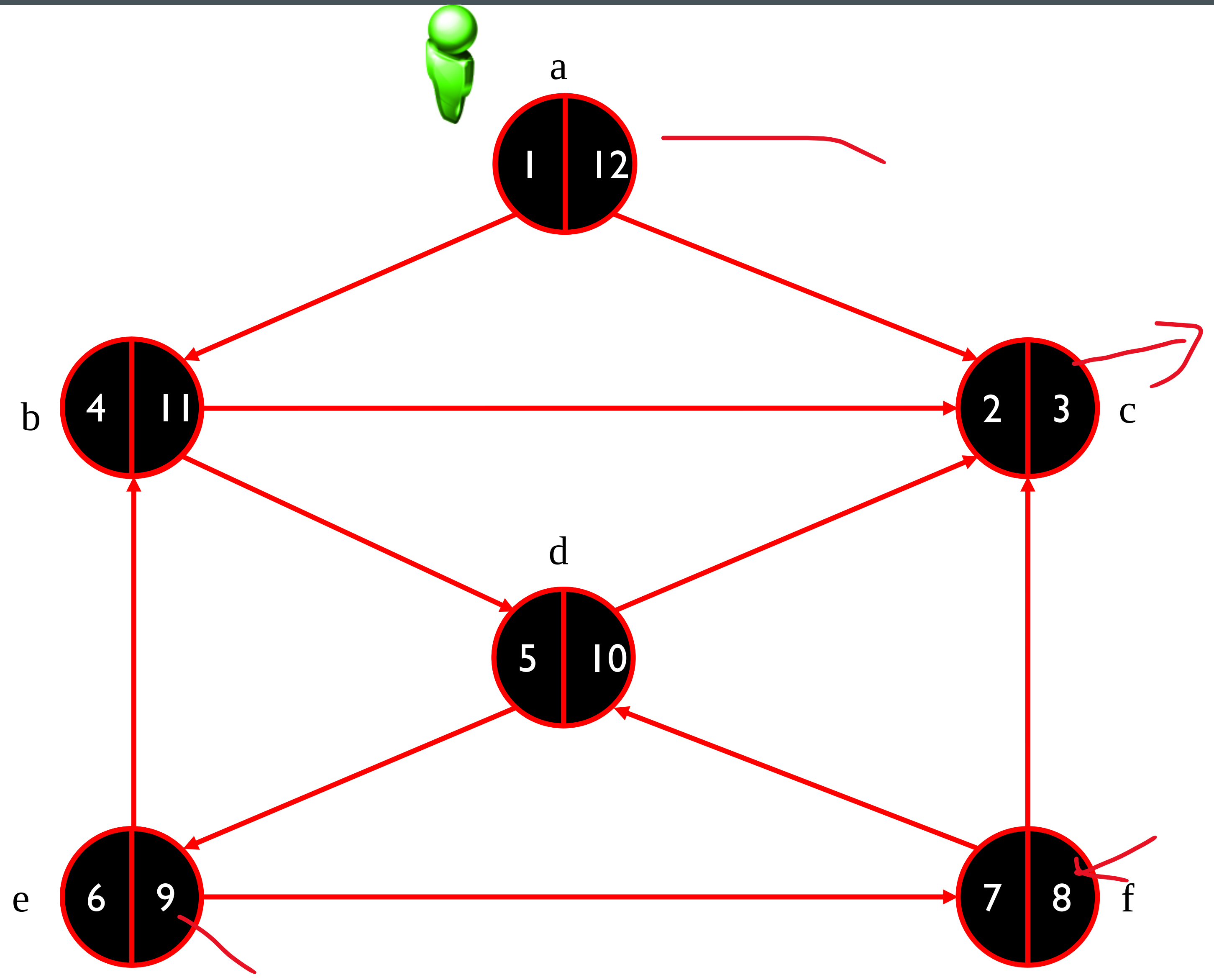
KOSARAJU-SHARIR ALGORITHM



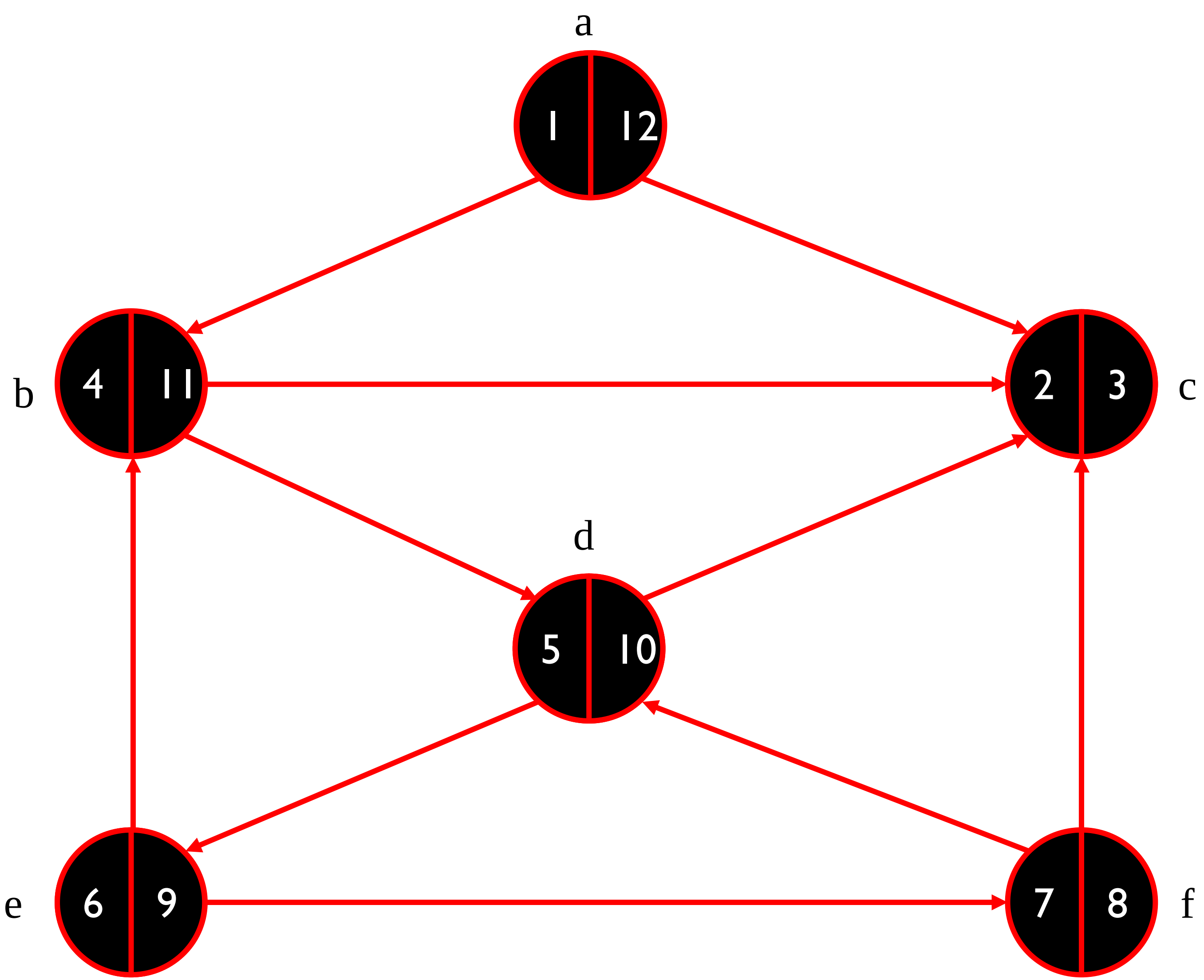
KOSARAJU-SHARIR ALGORITHM



KOSARAJU-SHARIR ALGORITHM

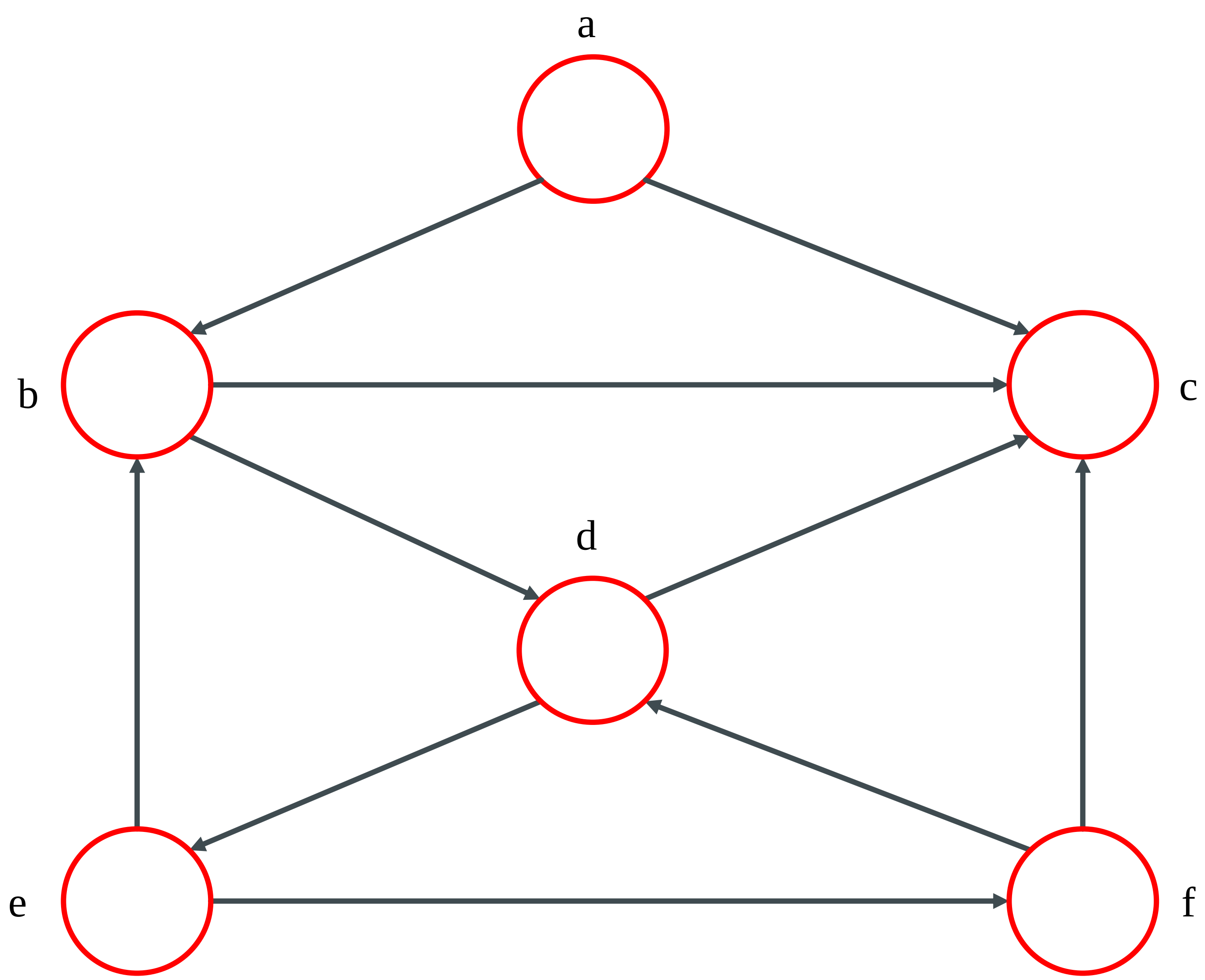


KOSARAJU-SHARIR ALGORITHM

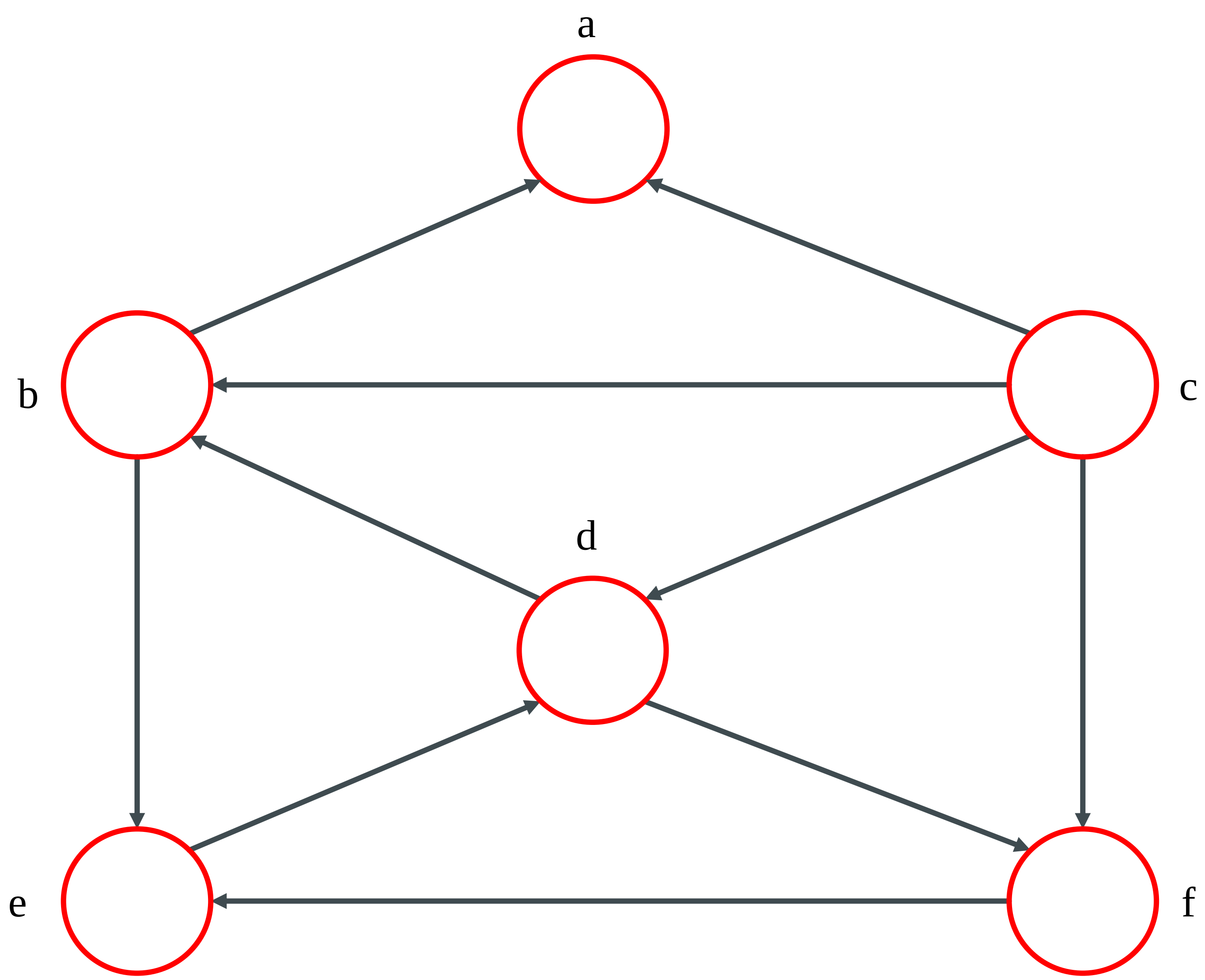


$$\{a, b, d, e, f, c\}$$

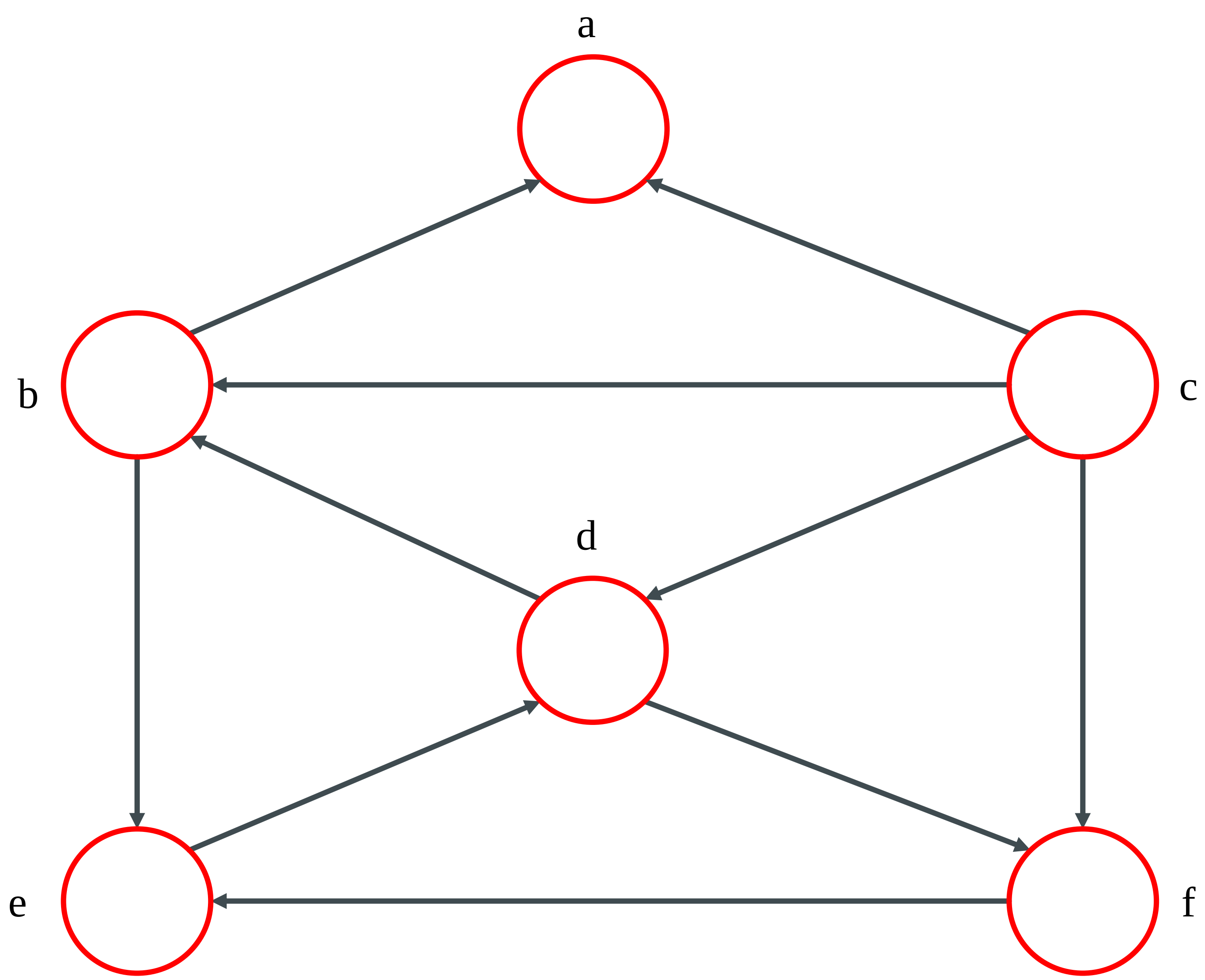
KOSARAJU-SHARIR ALGORITHM



KOSARAJU-SHARIR ALGORITHM



KOSARAJU-SHARIR ALGORITHM

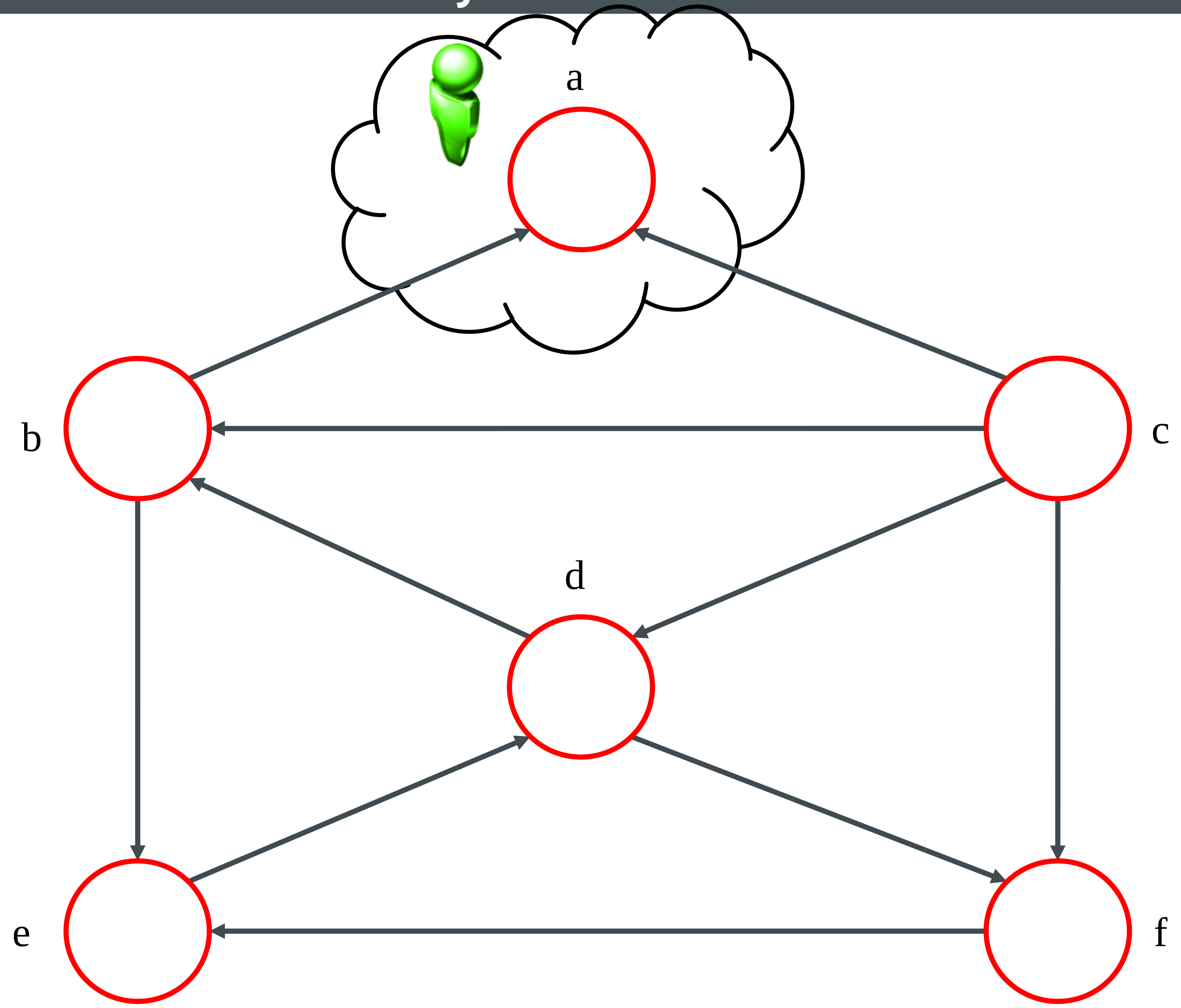


$\{a, b, d, e, f, c\}$

KOSARAJU-SHARIR ALGORITHM

SCCs

$\{a\}$

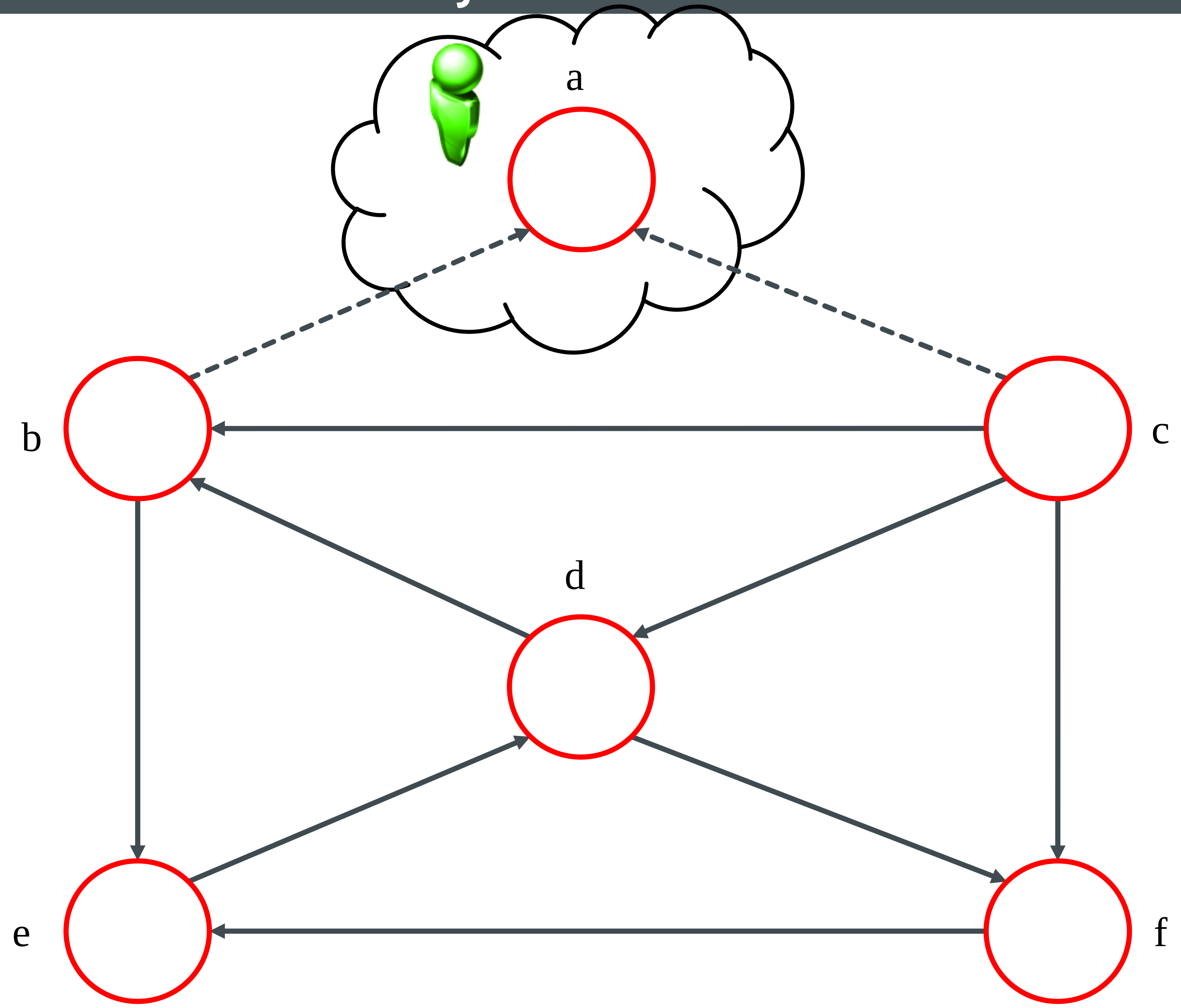


$\{a, b, d, e, f, c\}$

KOSARAJU-SHARIR ALGORITHM

SCCs

$\{a\}$



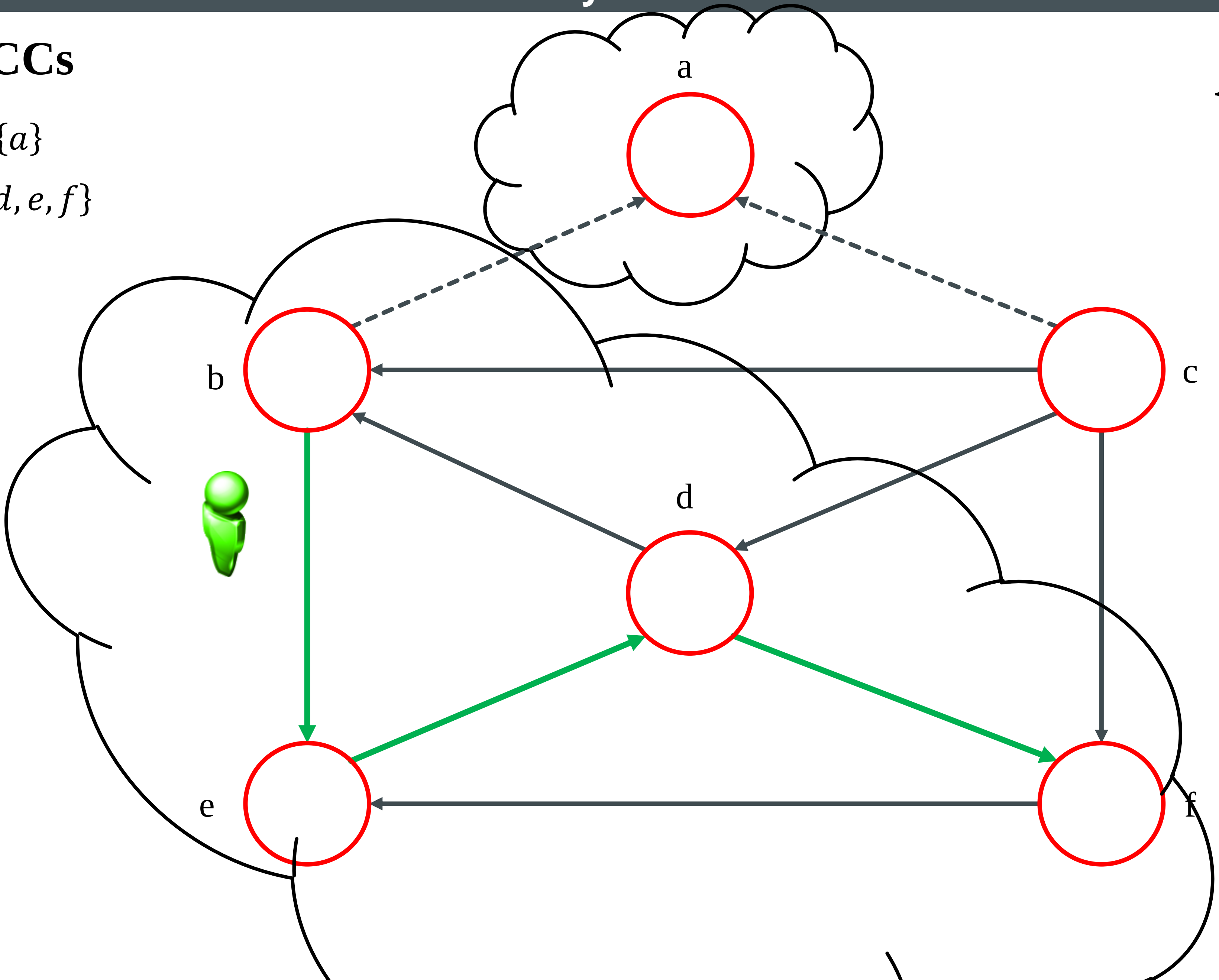
$\{a, b, d, e, f, c\}$

KOSARAJU-SHARIR ALGORITHM

SCCs

$\{a\}$

$\{b, d, e, f\}$



$\{a, b, d, e, f, c\}$

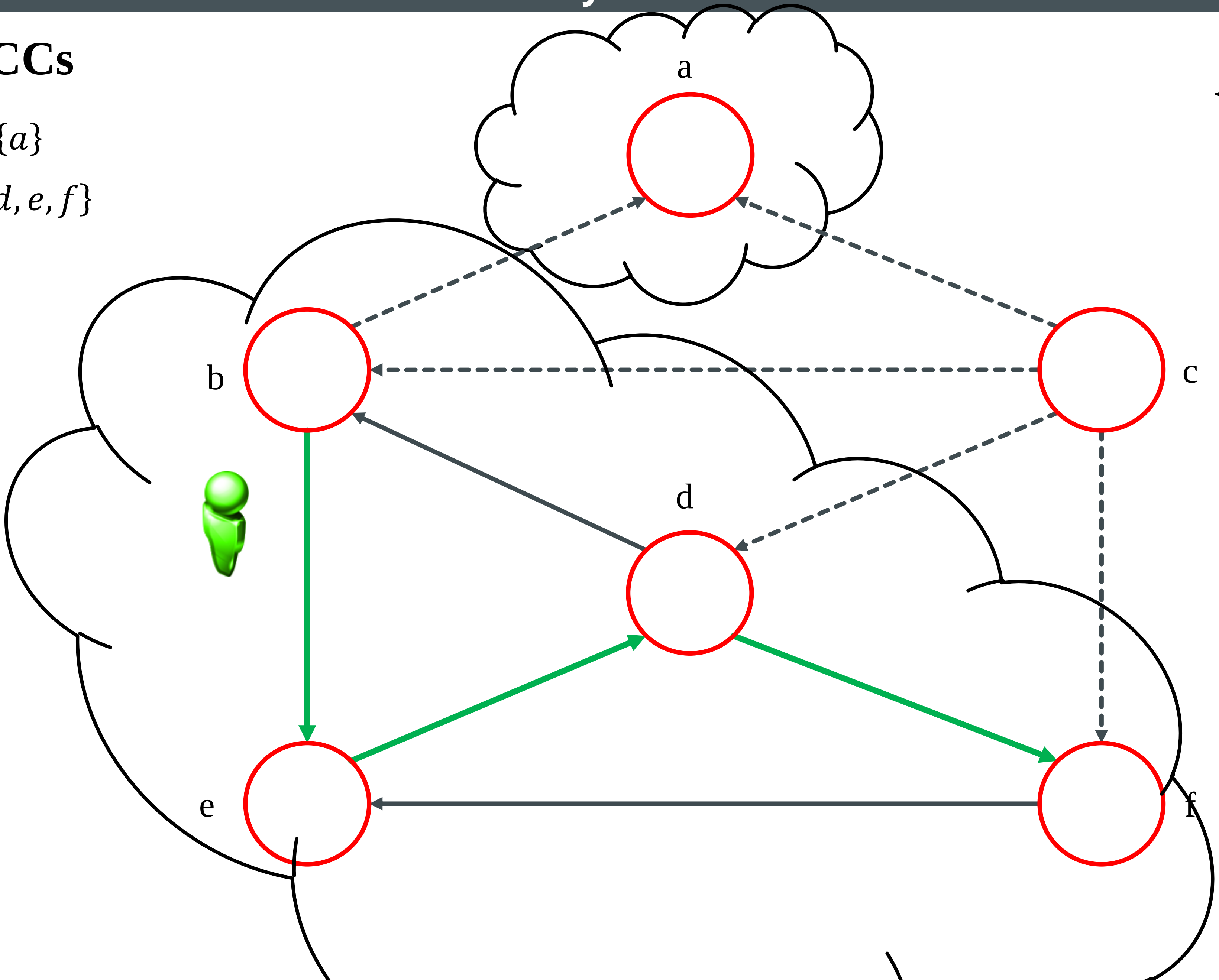
KOSARAJU-SHARIR ALGORITHM

SCCs

$\{a\}$

$\{b, d, e, f\}$

$\{a, b, d, e, f, c\}$



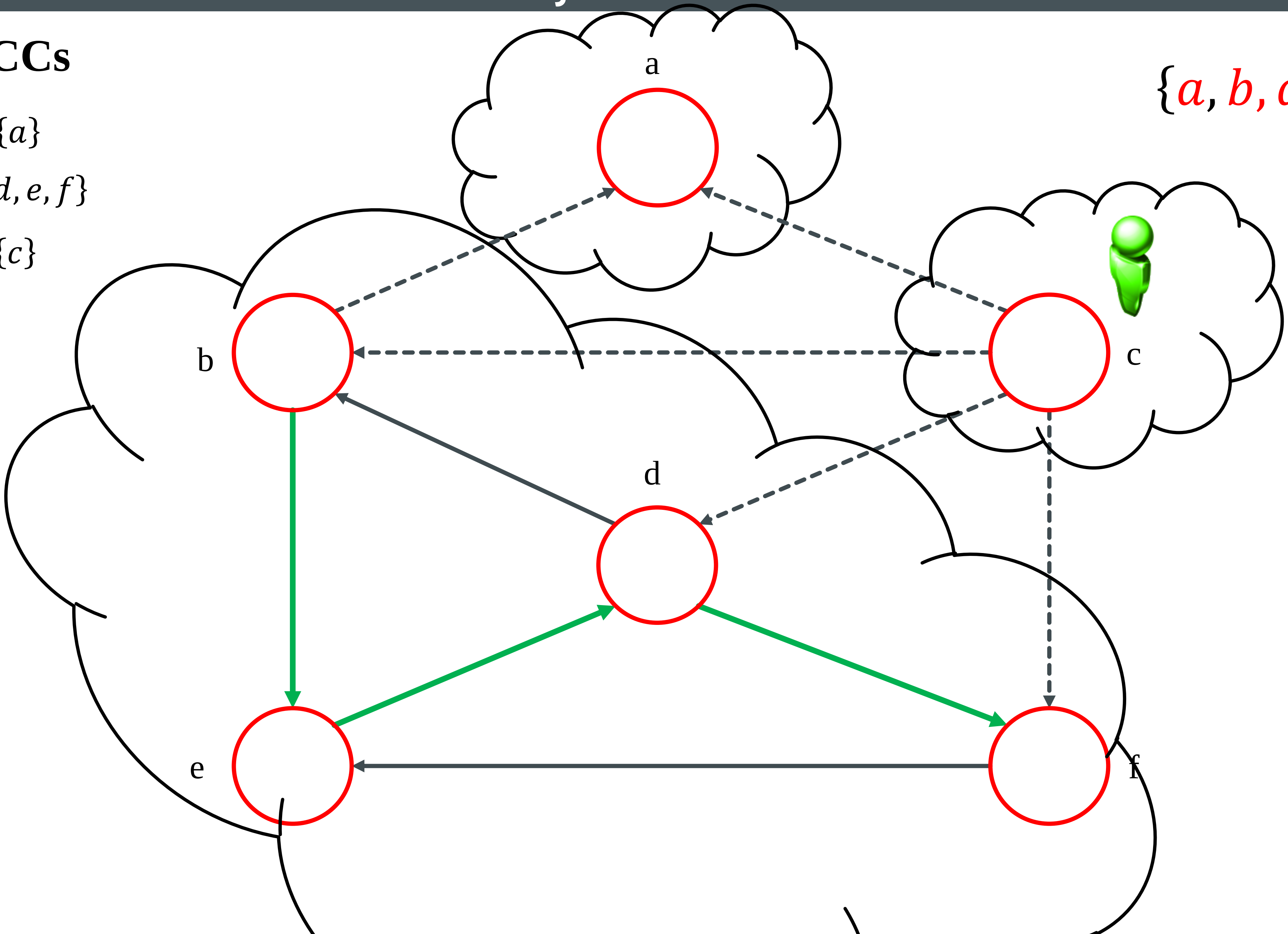
KOSARAJU-SHARIR ALGORITHM

SCCs

$\{a\}$

$\{b, d, e, f\}$

$\{c\}$



$\{a, b, d, e, f, c\}$

KOSARAJU-SHARIR ALGORITHM

Time Complexity

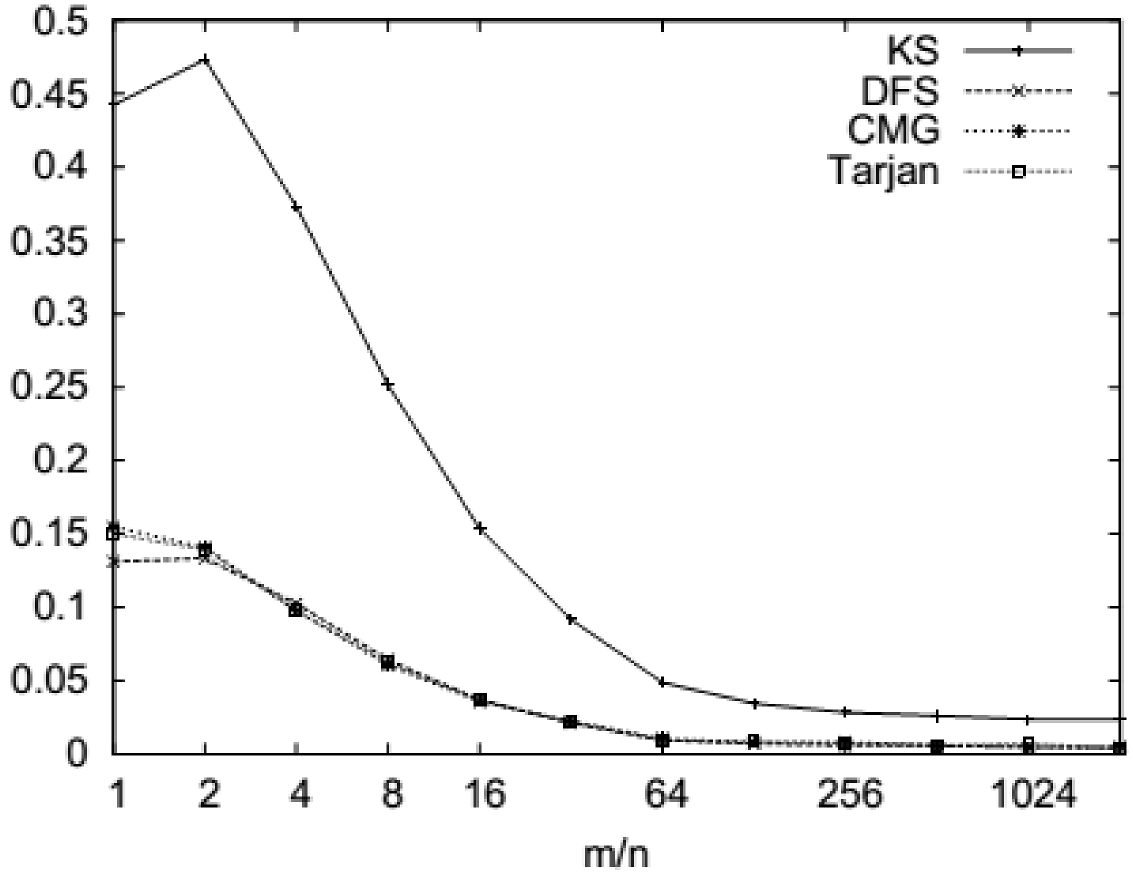
$$O(E+V)$$

Algorithm 3 Kosaraju-Sharir Algorithm

Input: A directed graph $G = (V, L)$
Output: Set of SCCs represented by SCC (initially empty)
Run DFS on Graph G to compute finishing time (reverse postorder) of each node.
$V^R = V$, $L^R = \emptyset$.
Build $G^R = (V^R, L^R)$
for $l = (u, v) \in L$ **do**
    $L^R = L^R \cup (v, u)$
Run DFS on G^R, considering vertices in the decreasing reverse postorder.
Output the vertices of each Depth First Traversal as an SCC.

Comparison

Tarjan's Alg.	Path-Based Alg.	Kosaraju-Sharir Alg.
$O(E+V)$	$O(E+V)$	$O(E+V)$



[1] H. N. Gabow, “Path-based depth-first search for strong and biconnected components,” *Information Processing Letters*, vol. 74, no. 3-4, pp. 107–114, 2000.

[2] M. Sharir, “A strong-connectivity algorithm and its applications in data flow analysis,” *Computers & Mathematics with Applications*, vol. 7, no. 1, pp. 67–72, 1981.

QUESTIONS/ANSWERS