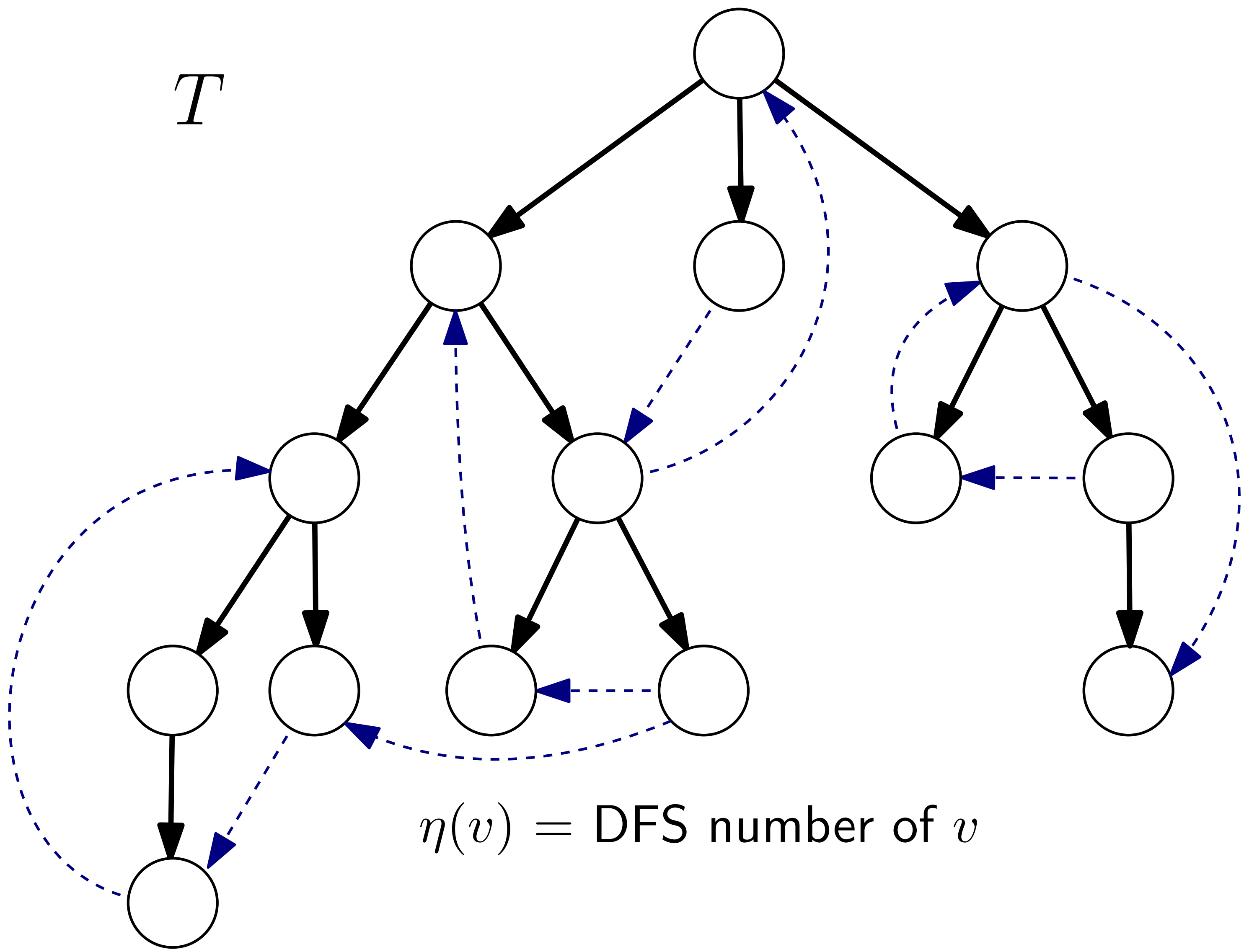
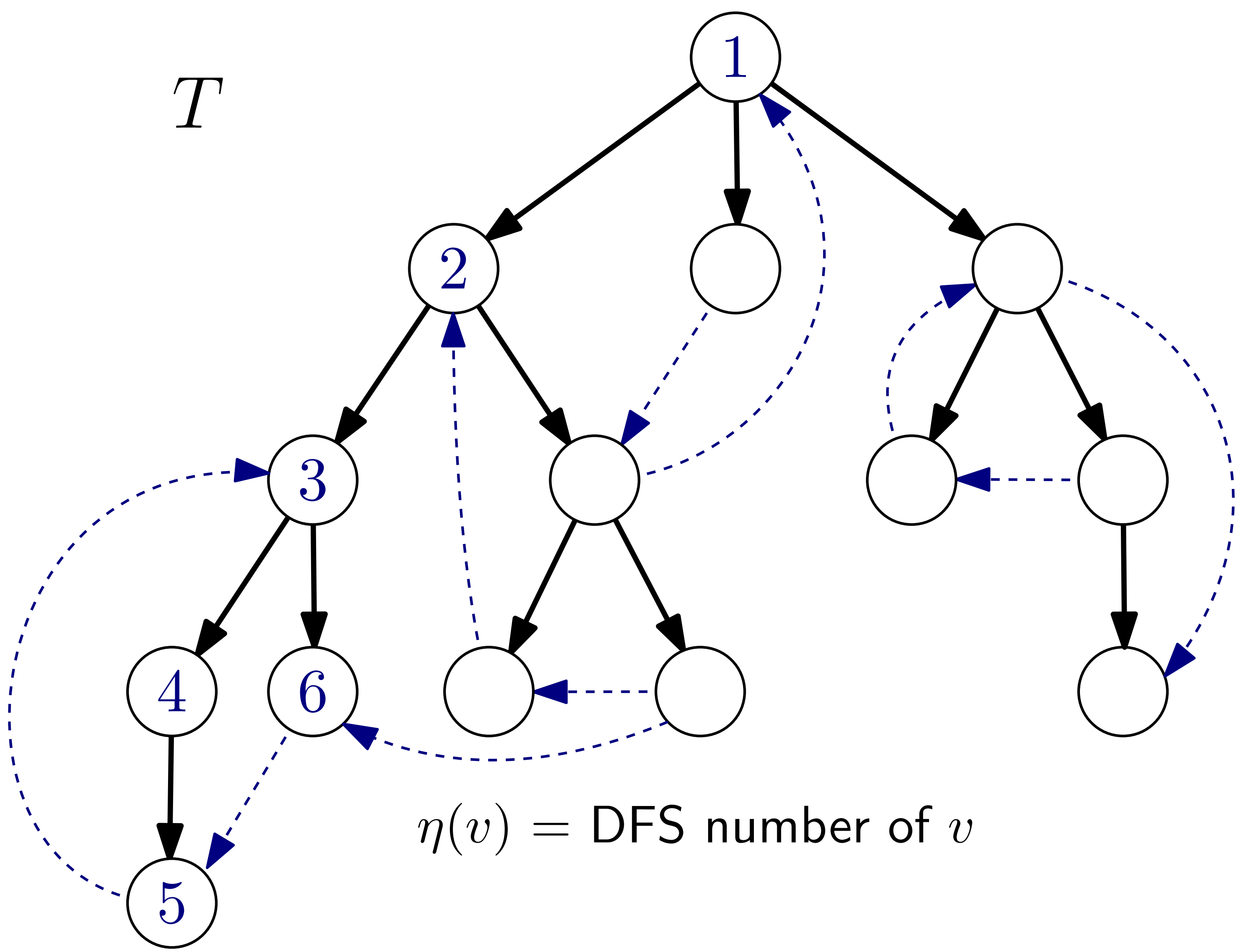
Strongly Connected Components

Tarjan's algorithm



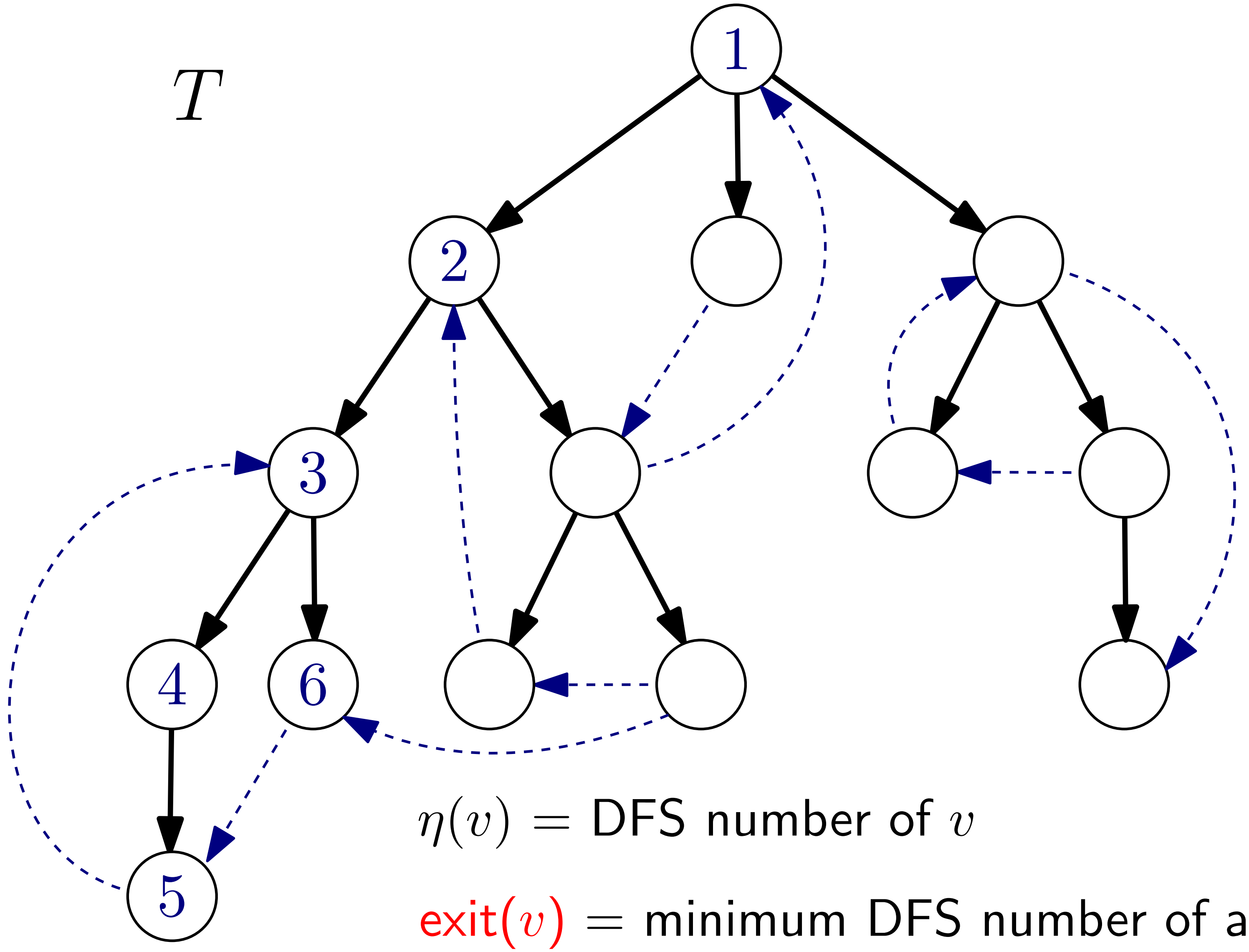
$\eta(v) =$ DFS number of v

Tarjan's algorithm



$\eta(v)$ = DFS number of v

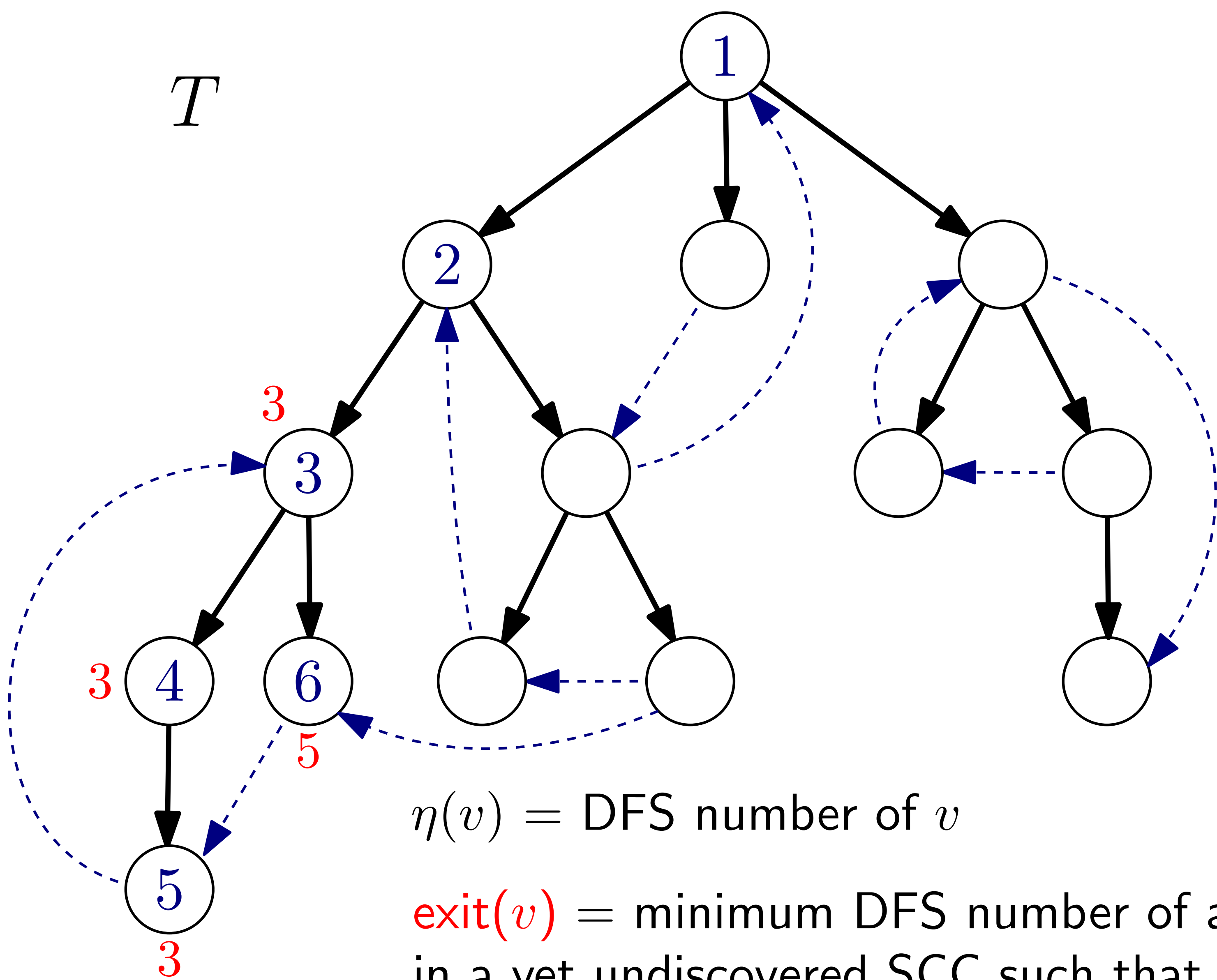
Tarjan's algorithm



$\eta(v)$ = DFS number of v

exit(v) = minimum DFS number of a vertex u in a yet undiscovered SCC such that u is reachable from v via a path in T followed by at most one final non-tree edge.

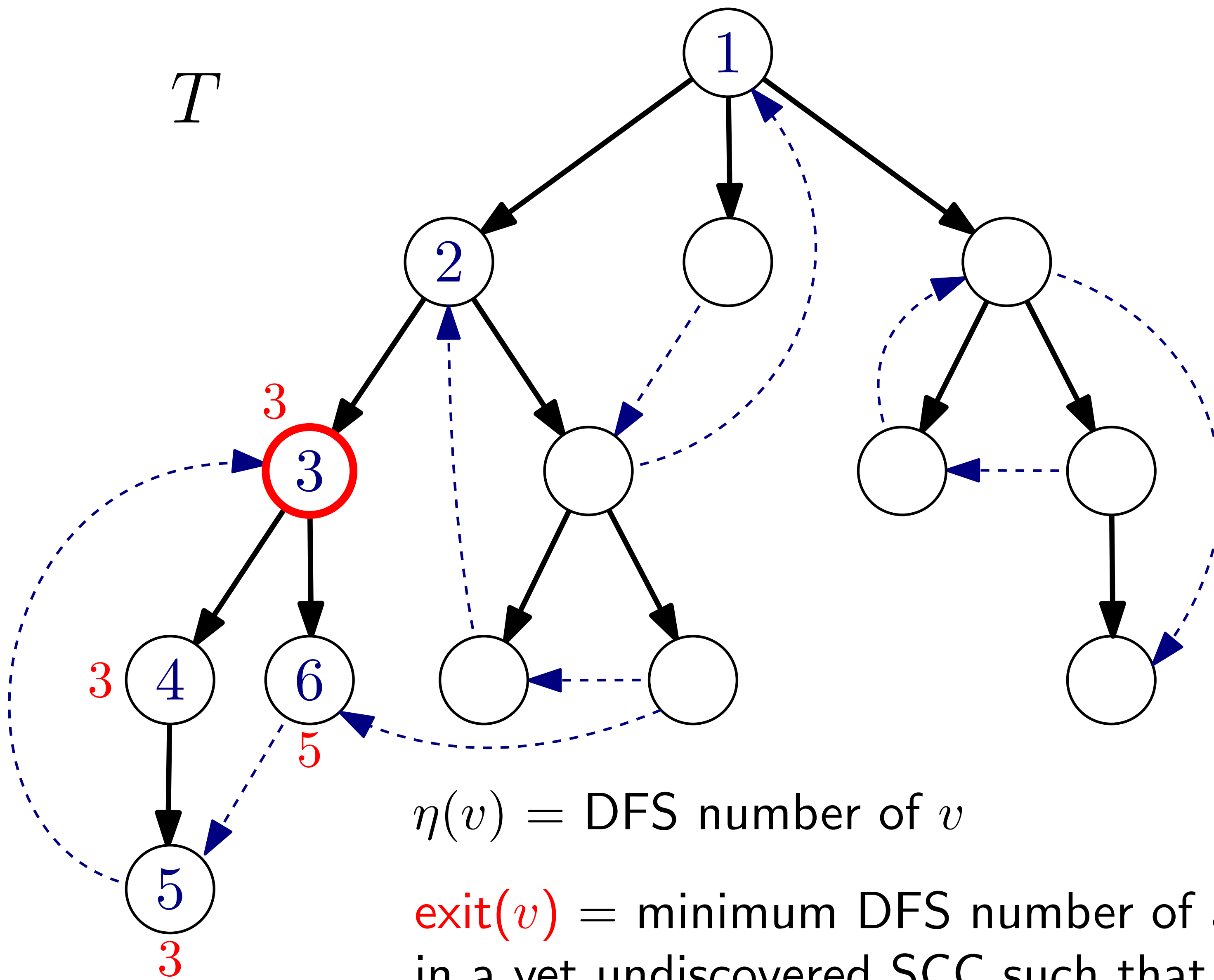
Tarjan's algorithm



$\eta(v)$ = DFS number of v

exit(v) = minimum DFS number of a vertex u in a yet undiscovered SCC such that u is reachable from v via a path in T followed by at most one final non-tree edge.

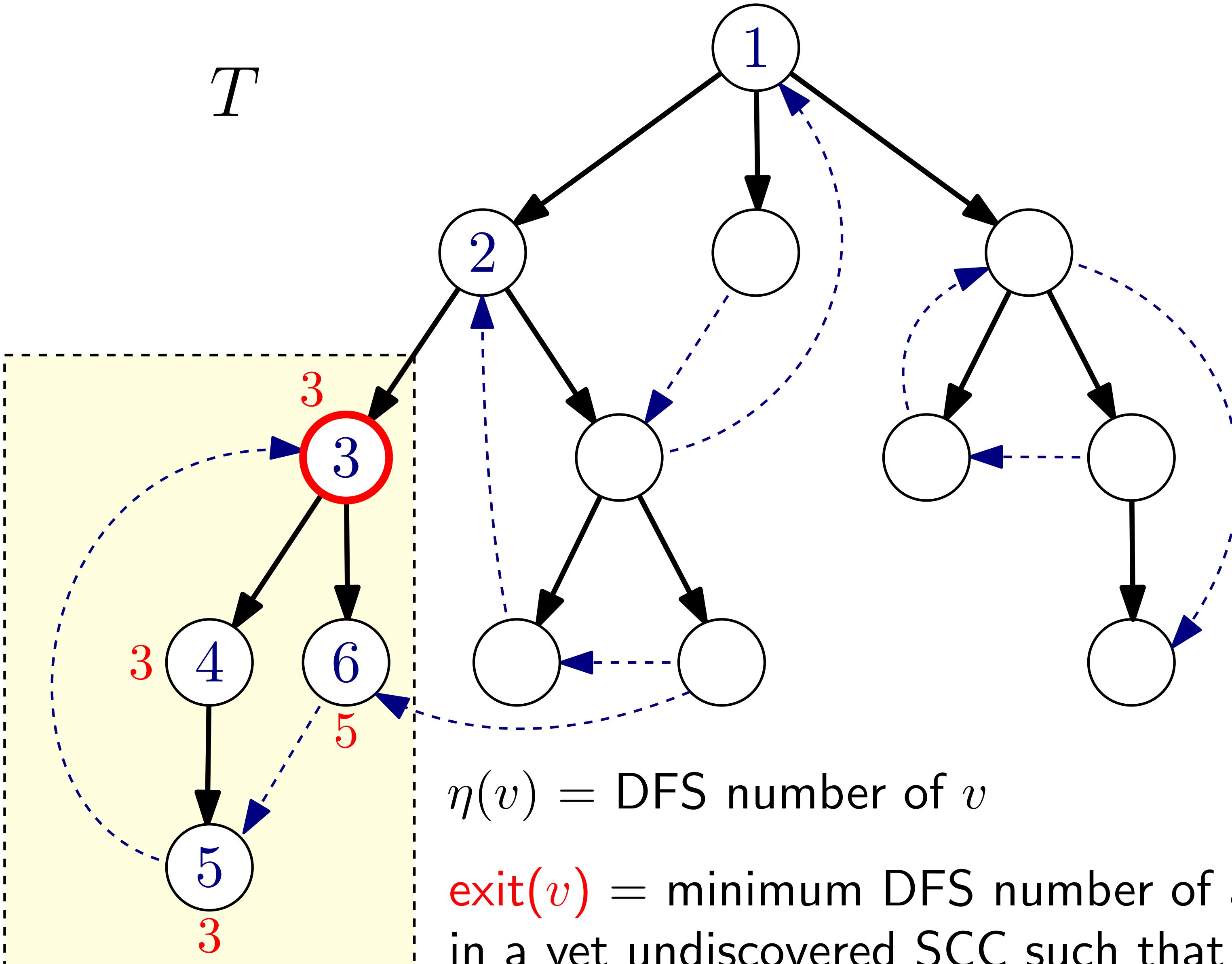
Tarjan's algorithm



$\eta(v)$ = DFS number of v

exit(v) = minimum DFS number of a vertex u in a yet undiscovered SCC such that u is reachable from v via a path in T followed by at most one final non-tree edge.

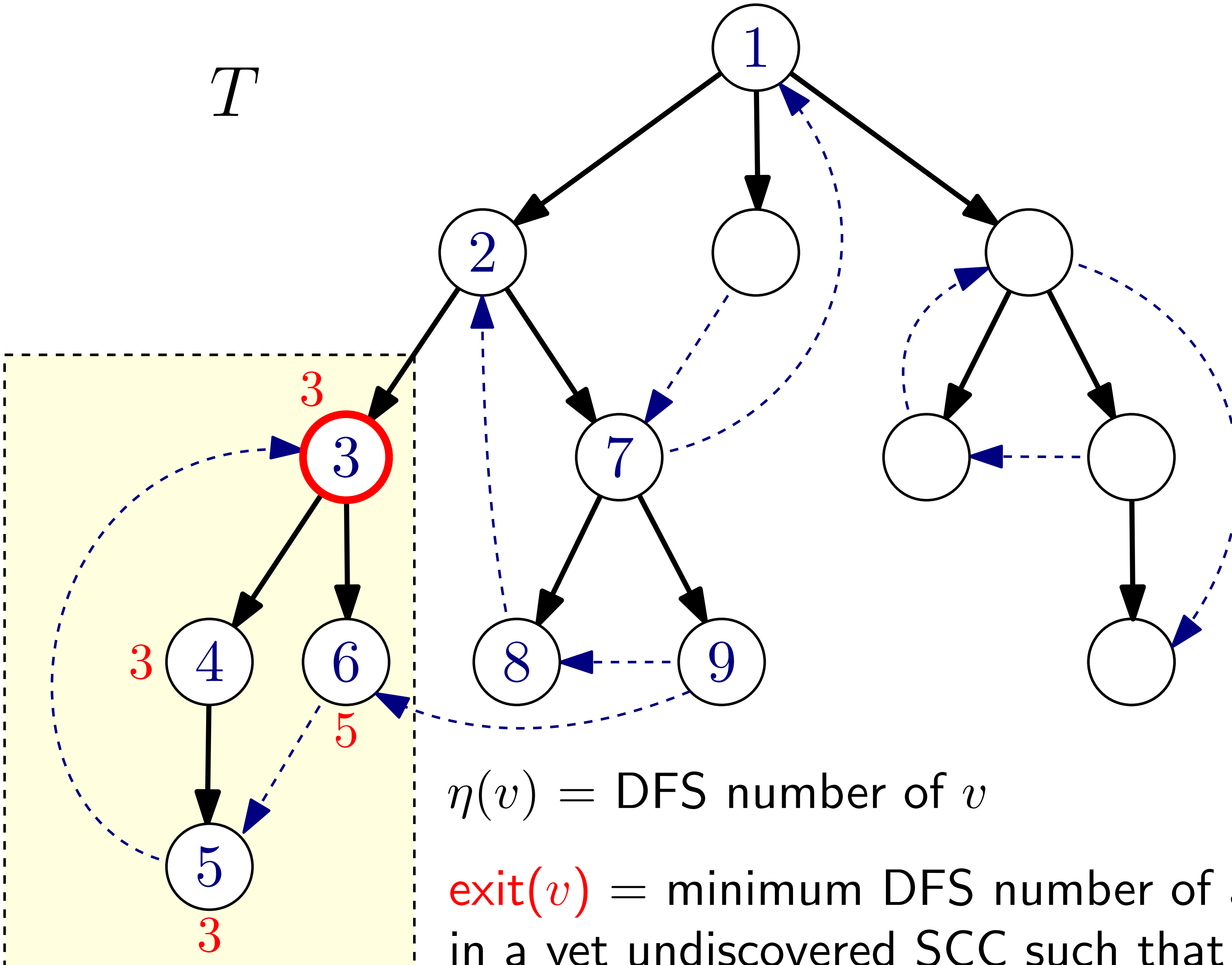
Tarjan's algorithm



$\eta(v)$ = DFS number of v

exit(v) = minimum DFS number of a vertex u in a yet undiscovered SCC such that u is reachable from v via a path in T followed by at most one final non-tree edge.

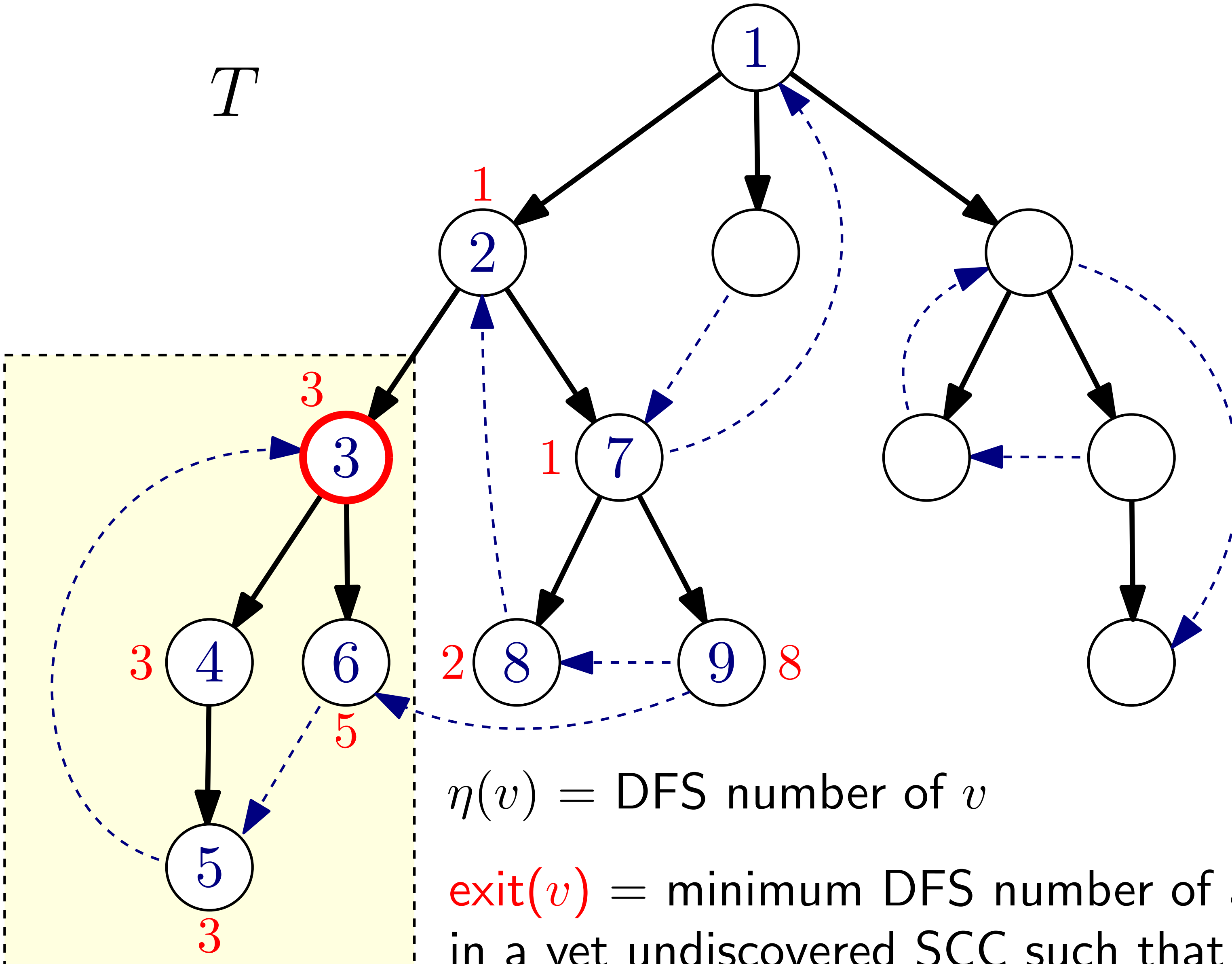
Tarjan's algorithm



$\eta(v)$ = DFS number of v

exit(v) = minimum DFS number of a vertex u in a yet undiscovered SCC such that u is reachable from v via a path in T followed by at most one final non-tree edge.

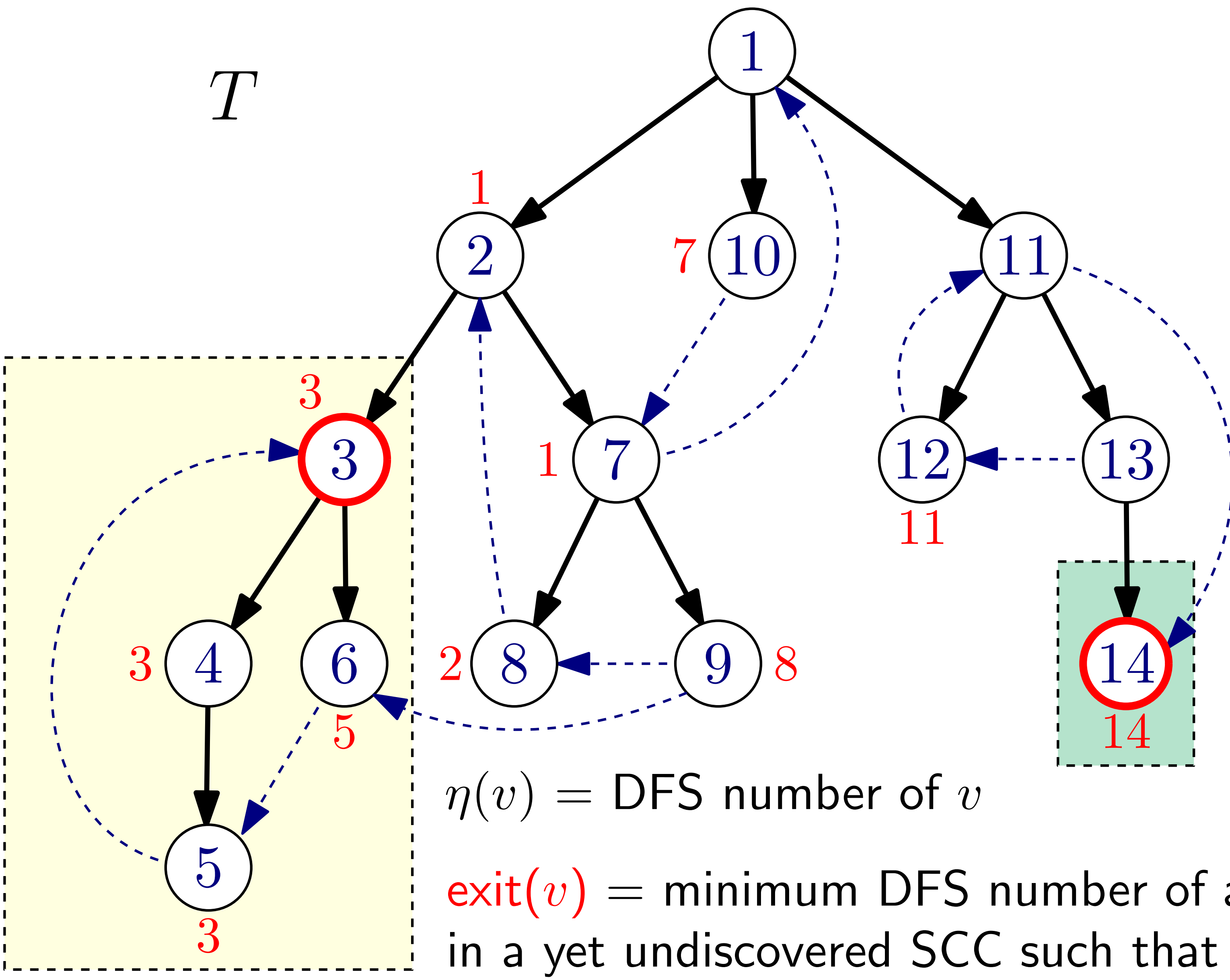
Tarjan's algorithm



$\eta(v)$ = DFS number of v

exit(v) = minimum DFS number of a vertex u in a yet undiscovered SCC such that u is reachable from v via a path in T followed by at most one final non-tree edge.

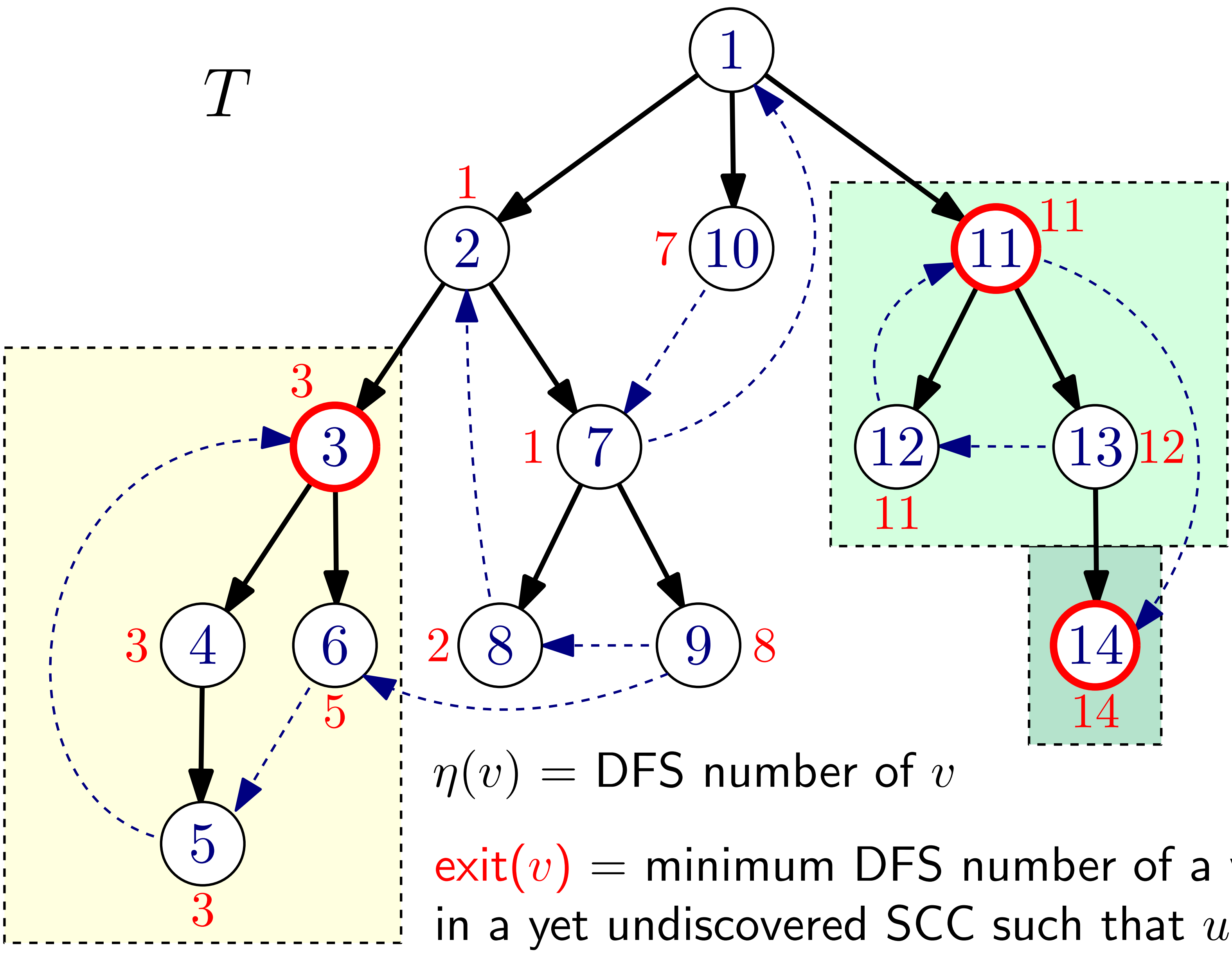
Tarjan's algorithm



$\eta(v)$ = DFS number of v

exit(v) = minimum DFS number of a vertex u in a yet undiscovered SCC such that u is reachable from v via a path in T followed by at most one final non-tree edge.

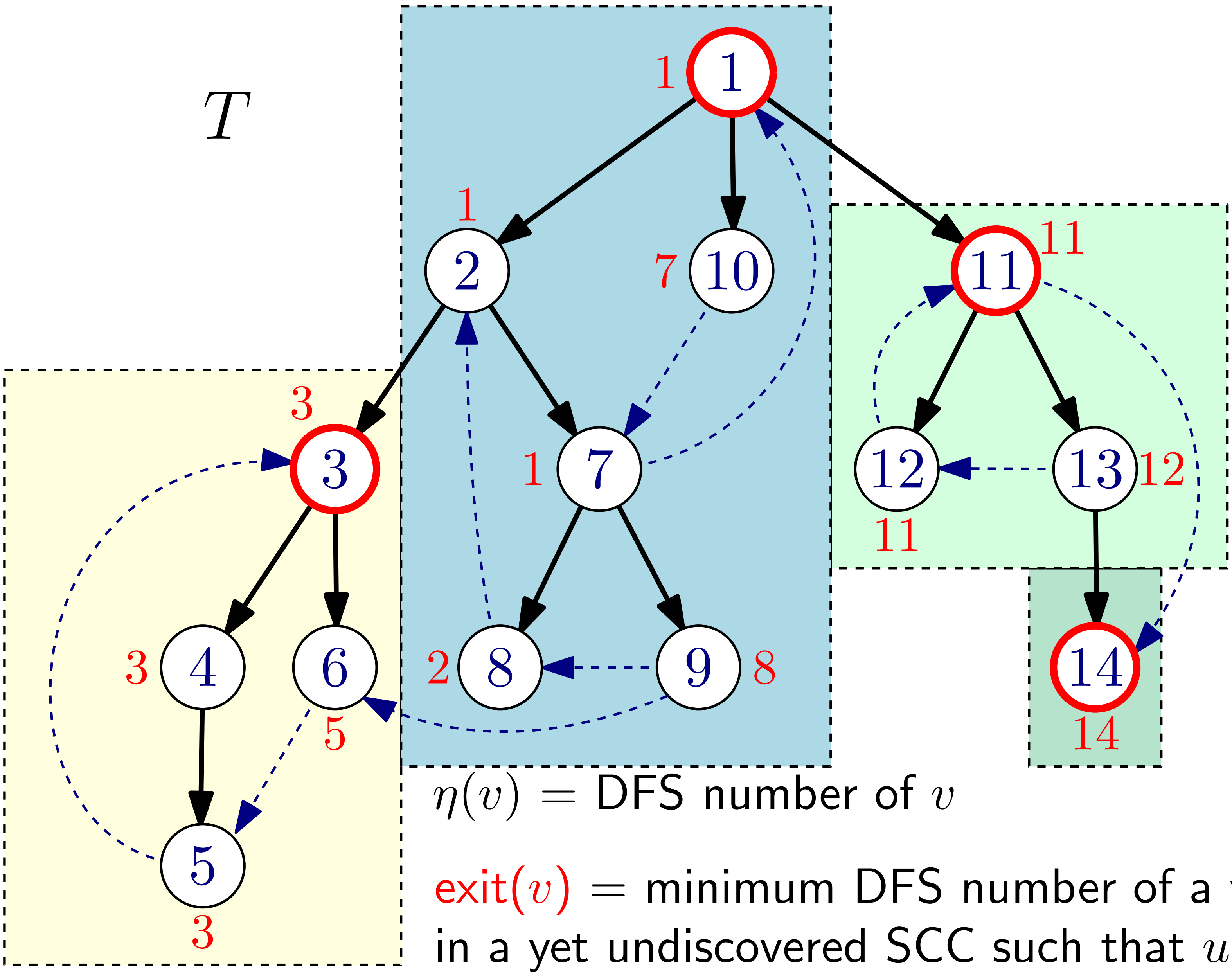
Tarjan's algorithm



$\eta(v)$ = DFS number of v

exit(v) = minimum DFS number of a vertex u in a yet undiscovered SCC such that u is reachable from v via a path in T followed by at most one final non-tree edge.

Tarjan's algorithm



$\eta(v)$ = DFS number of v

exit(v) = minimum DFS number of a vertex u in a yet undiscovered SCC such that u is reachable from v via a path in T followed by at most one final non-tree edge.

Proof of correctness

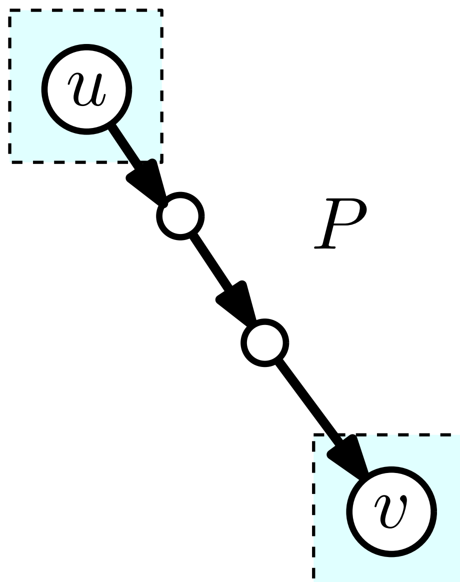
Claim: Let C be a SCC. The subgraph $T[C]$ of T induced by C is connected.

Proof:

Let u be the first vertex of C that is visited by the algorithm.

Let $v \in C$, with $v \neq u$.

- u must be an ancestor of v in T (by the properties of DFS).



Proof of correctness

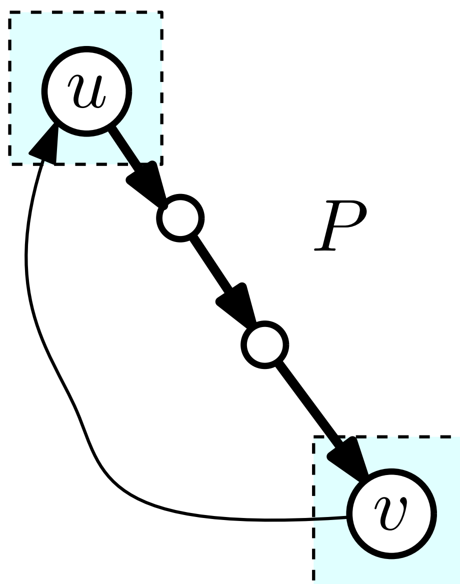
Claim: Let C be a SCC. The subgraph $T[C]$ of T induced by C is connected.

Proof:

Let u be the first vertex of C that is visited by the algorithm.

Let $v \in C$, with $v \neq u$.

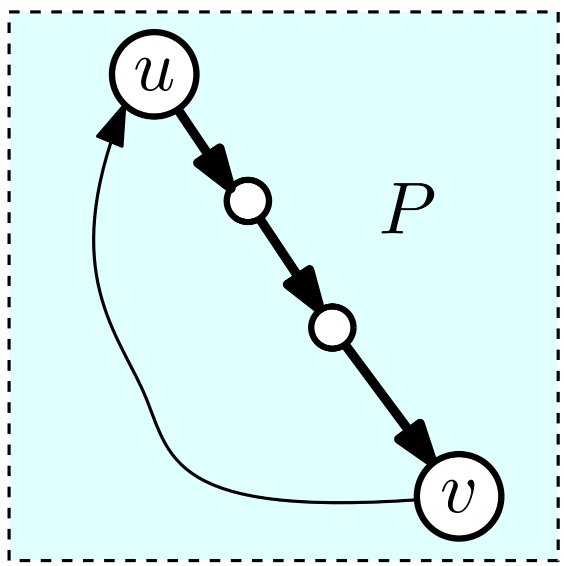
- u must be an ancestor of v in T (by the properties of DFS).



Proof of correctness

Claim: Let C be a SCC. The subgraph $T[C]$ of T induced by C is connected.
Proof:
Let u be the first vertex of C that is visited by the algorithm.
Let $v \in C$, with $v \neq u$.

- u must be an ancestor of v in T (by the properties of DFS).



Proof of correctness

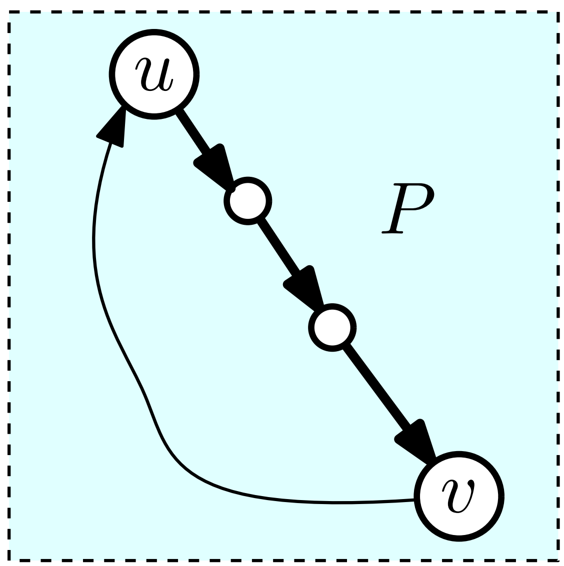
Claim: Let C be a SCC. The subgraph $T[C]$ of T induced by C is connected.

Proof:

Let u be the first vertex of C that is visited by the algorithm.

Let $v \in C$, with $v \neq u$.

- u must be an ancestor of v in T (by the properties of DFS).



- There is a path from u to v in $G \implies$ the vertices in P are in $C \implies u$ and v must also be connected in $T[C]$.

□

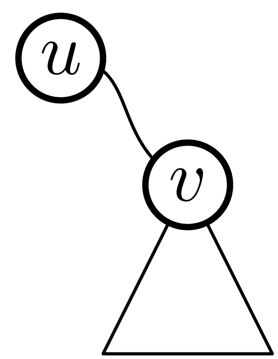
Proof of correctness

Definition: the *head* u of a SCC C is the (unique!) vertex of C having minimum depth in T.

Proof of correctness

Definition: the *head* u of a SCC C is the (unique!) vertex of C having minimum depth in T.

Claim: $\forall v \in C \setminus \{u\}$, $\eta(v) \neq exit(v)$.

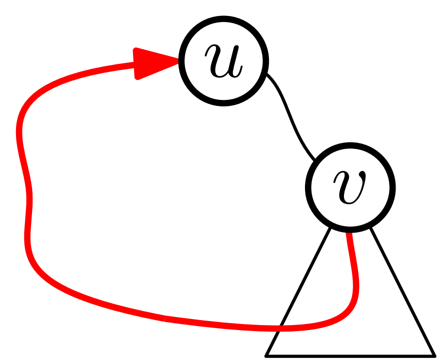


Proof of correctness

Definition: the *head* u of a SCC C is the (unique!) vertex of C having minimum depth in T.

Claim: $\forall v \in C \setminus \{u\}$, $\eta(v) \neq exit(v)$.

- There is a path P from v to u.

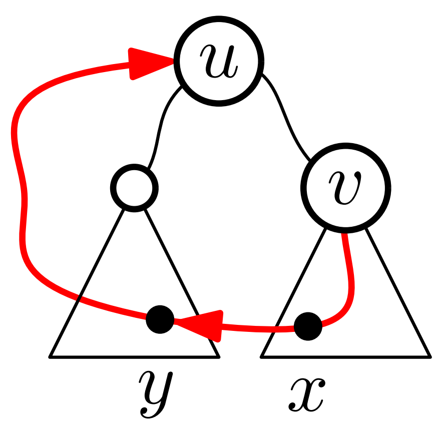


Proof of correctness

Definition: the *head* u of a SCC C is the (unique!) vertex of C having minimum depth in T.

Claim: $\forall v \in C \setminus \{u\}$, $\eta(v) \neq exit(v)$.

- There is a path P from v to u.
- Consider the first edge (x, y) of P such that $y \notin T_v$.

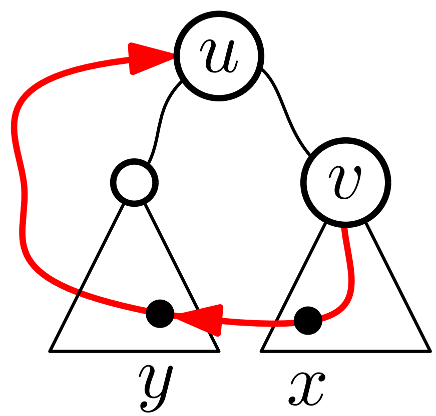


Proof of correctness

Definition: the *head* u of a SCC C is the (unique!) vertex of C having minimum depth in T.

Claim: $\forall v \in C \setminus \{u\}$, $\eta(v) \neq exit(v)$.

- There is a path P from v to u.
- Consider the first edge (x, y) of P such that $y \notin T_v$.
- y is visited before v in the DFS.

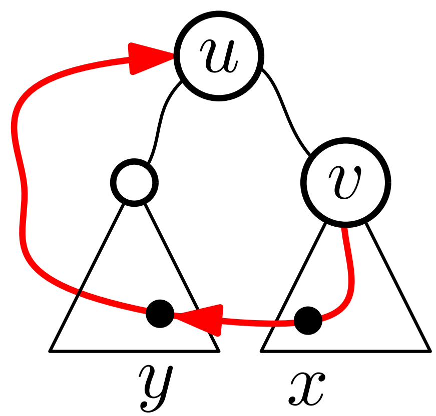


Proof of correctness

Definition: the *head* u of a SCC C is the (unique!) vertex of C having minimum depth in T.

Claim: $\forall v \in C \setminus \{u\}$, $\eta(v) \neq exit(v)$.

- There is a path P from v to u.
- Consider the first edge (x, y) of P such that $y \notin T_v$.
- y is visited before v in the DFS.
- $exit(v) \leq \eta(y) < \eta(v)$.



$\square$

Proof of correctness

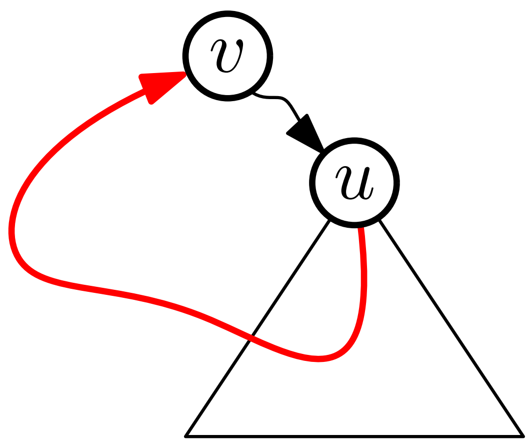
Claim: Let u be the first encountered head in postorder. $\eta(u) = exit(u)$.

- Assume that there is a vertex v s.t. $\eta(v) = exit(u) < \eta(u)$.

Proof of correctness

Claim: Let u be the first encountered head in postorder. $\eta(u) = exit(u)$.

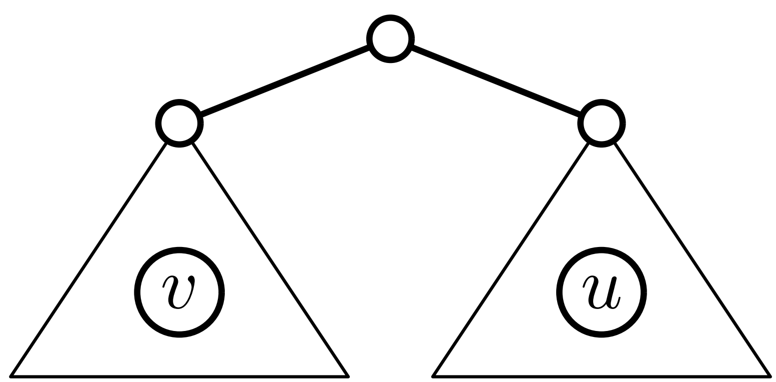
- Assume that there is a vertex v s.t. $\eta(v) = exit(u) < \eta(u)$.
- v cannot be an ancestor of u (otherwise $v \in C$ and u is not the head of C).



Proof of correctness

Claim: Let u be the first encountered head in postorder. $\eta(u) = exit(u)$.

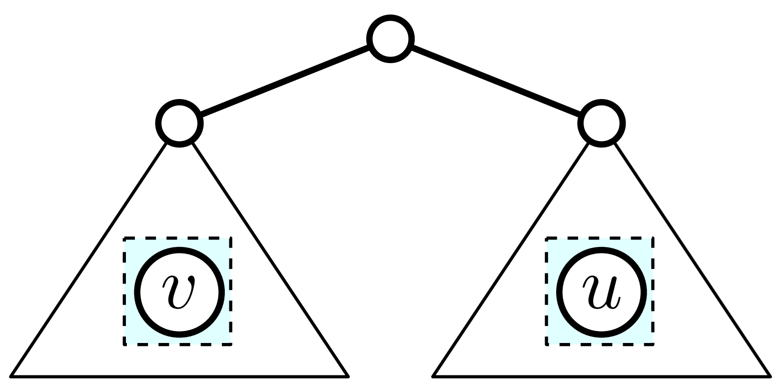
- Assume that there is a vertex v s.t. $\eta(v) = exit(u) < \eta(u)$.
- v cannot be an ancestor of u (otherwise $v \in C$ and u is not the head of C).



Proof of correctness

Claim: Let u be the first encountered head in postorder. $\eta(u) = exit(u)$.

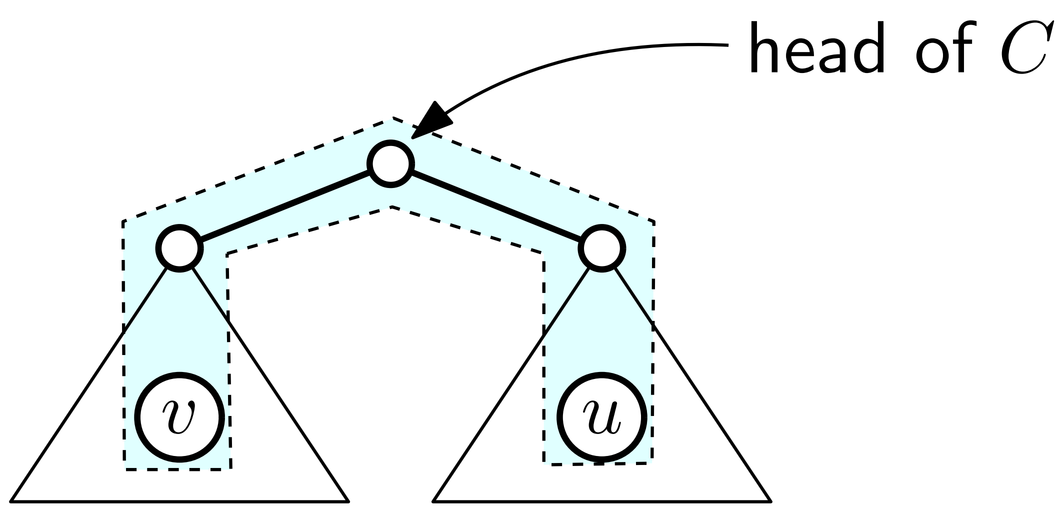
- Assume that there is a vertex v s.t. $\eta(v) = exit(u) < \eta(u)$.
- v cannot be an ancestor of u (otherwise $v \in C$ and u is not the head of C).
- If $v \in C$, then u and v are connected in $T[C] \implies$ the lowest common ancestor of u and v is in C.



Proof of correctness

Claim: Let u be the first encountered head in postorder.
$\eta(u) = exit(u)$.

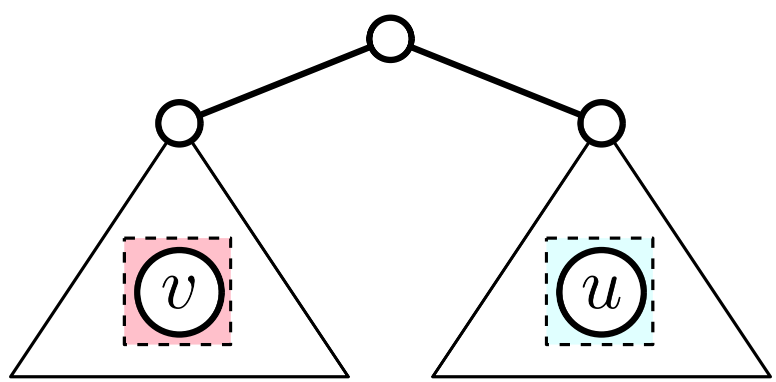
- Assume that there is a vertex v s.t. $\eta(v) = exit(u) < \eta(u)$.
- v cannot be an ancestor of u (otherwise $v \in C$ and u is not the head of C).
- If $v \in C$, then u and v are connected in $T[C] \implies$ the lowest common ancestor of u and v is in C. ↯



Proof of correctness

Claim: Let u be the first encountered head in postorder. $\eta(u) = exit(u)$.

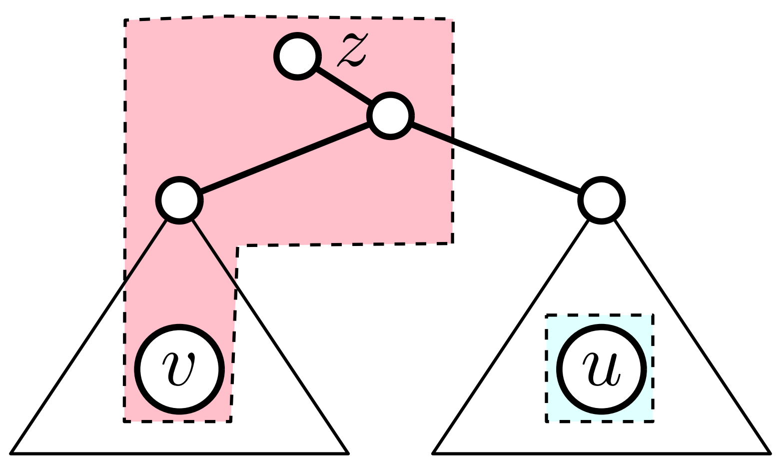
- Assume that there is a vertex v s.t. $\eta(v) = exit(u) < \eta(u)$.
- v cannot be an ancestor of u (otherwise $v \in C$ and u is not the head of C).
- If $v \in C$, then u and v are connected in $T[C] \implies$ the lowest common ancestor of u and v is in C. ↯
- If $v \in C' \neq C$ then the head z of C' must be an ancestor of $u \implies$ there is a path from u to z and vice-versa.



Proof of correctness

Claim: Let u be the first encountered head in postorder. $\eta(u) = exit(u)$.

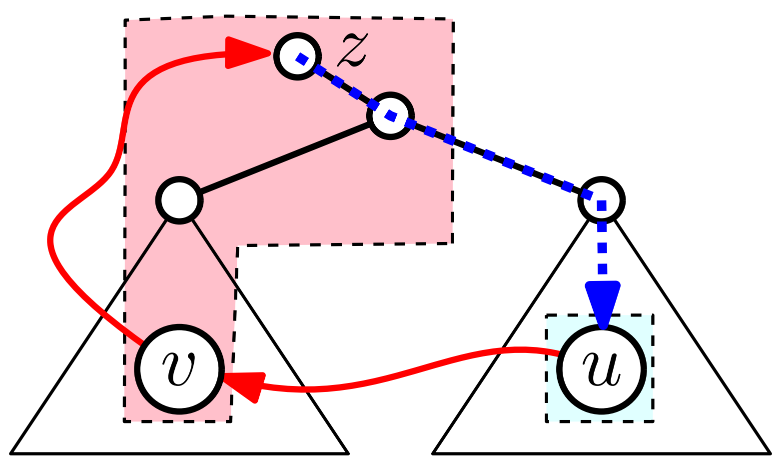
- Assume that there is a vertex v s.t. $\eta(v) = exit(u) < \eta(u)$.
- v cannot be an ancestor of u (otherwise $v \in C$ and u is not the head of C).
- If $v \in C$, then u and v are connected in $T[C] \implies$ the lowest common ancestor of u and v is in C. ↯
- If $v \in C' \neq C$ then the head z of C' must be an ancestor of $u \implies$ there is a path from u to z and vice-versa.



Proof of correctness

Claim: Let u be the first encountered head in postorder. $\eta(u) = exit(u)$.

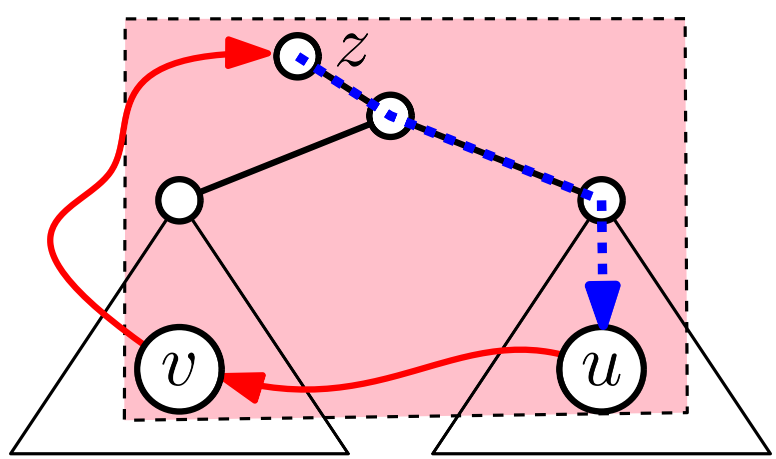
- Assume that there is a vertex v s.t. $\eta(v) = exit(u) < \eta(u)$.
- v cannot be an ancestor of u (otherwise $v \in C$ and u is not the head of C).
- If $v \in C$, then u and v are connected in $T[C] \implies$ the lowest common ancestor of u and v is in C. ↯
- If $v \in C' \neq C$ then the head z of C' must be an ancestor of $u \implies$ there is a path from u to z and vice-versa.



Proof of correctness

Claim: Let u be the first encountered head in postorder. $\eta(u) = exit(u)$.

- Assume that there is a vertex v s.t. $\eta(v) = exit(u) < \eta(u)$.
- v cannot be an ancestor of u (otherwise $v \in C$ and u is not the head of C).
- If $v \in C$, then u and v are connected in $T[C] \implies$ the lowest common ancestor of u and v is in C. ↯
- If $v \in C' \neq C$ then the head z of C' must be an ancestor of $u \implies$ there is a path from u to z and vice-versa. ↯



The Algorithm

While $\exists$ vertex $u \in G$ (that has not been deleted):

- cnt $\leftarrow 0$; $T \leftarrow (\{u\}, \emptyset)$
- SCC(u)

SCC(u):

- $\eta(u) \leftarrow$ cnt; cnt $\leftarrow$ cnt $+1$; $exit(u) \leftarrow \eta(u)$
- For each $(u, v) \in E$:
  - If v has not yet been visited:
    - Add (u, v) to T
    - SCC(v)
    - $exit(u) \leftarrow \min\{exit(u), exit(v)\}$
  - Else:
    - $exit(u) \leftarrow \min\{exit(u), \eta(v)\}$
- If $exit(u) = \eta(u)$:
  - Report a new SCC C containing all the descendants of u in T
  - Delete the vertices in C from G and T

    (vertices can be "deleted" in constant time by marking them)

The Algorithm

While $\exists$ vertex $u \in G$ (that has not been deleted):

- cnt $\leftarrow 0$; $T \leftarrow (\{u\}, \emptyset)$
- SCC(u)

SCC(u):

- $\eta(u) \leftarrow$ cnt; cnt $\leftarrow$ cnt $+1$; $exit(u) \leftarrow \eta(u)$
- For each $(u, v) \in E$:
  - If v has not yet been visited:
    - Add (u, v) to T
    - SCC(v)
    - $exit(u) \leftarrow \min\{exit(u), exit(v)\}$
  - Else:
    - $exit(u) \leftarrow \min\{exit(u), \eta(v)\}$
- If $exit(u) = \eta(u)$:
  - Report a new SCC C containing all the descendants of u in T
  - Delete the vertices in C from G and T

    (vertices can be "deleted" in constant time by marking them)

The Algorithm

While $\exists$ vertex $u \in G$ (that has not been deleted):

- cnt $\leftarrow 0$; $T \leftarrow (\{u\}, \emptyset)$ $S \leftarrow$ Empty stack
- SCC(u)

SCC(u):

- $\eta(u) \leftarrow$ cnt; cnt $\leftarrow$ cnt $+1$; $exit(u) \leftarrow \eta(u)$ Push u into S
- For each $(u, v) \in E$:
  - If v has not yet been visited:
    - Add (u, v) to T
    - SCC(v)
    - $exit(u) \leftarrow \min\{exit(u), exit(v)\}$
  - Else:
    - $exit(u) \leftarrow \min\{exit(u), \eta(v)\}$
- If $exit(u) = \eta(u)$:
  - $C = \emptyset$; do $z \leftarrow$ Pop from S; $C \leftarrow C \cup \{z\}$ while $z \neq u$;
  - Delete the vertices in C from G and T

    (vertices can be "deleted" in constant time by marking them)