2-SAT and Strongly Connected Components

2-SAT

Input: A formula ϕ consisting of

- A set of n boolean variables $x_1, \ldots, x_n$
- A conjuction of m clauses $C_1, \ldots, C_m$, i.e., disjunctions of 2 literals $C_j = (c_j^{(1)} \vee c_j^{(2)})$, where a literal is either a variable or its negation.

A truth assignment is a function $\tau : \{x_1, \ldots, x_n\} \to \{\top, \bot\}$

- A clause $C_j = (c_j^{(1)} \vee c_j^{(2)})$ is *satisfied* by τ according to the rules of boolean algebra.
- ϕ is satisfied iff all m clauses $C_1, \ldots, C_m$ are satisfied.

Question: Is there a truth assigment that satisfies ϕ?

Example

Formula

$$\phi = (x_1 \vee \overline{x_2}) \wedge (x_2 \vee x_4) \wedge (\overline{x_1} \vee \overline{x_3}) \wedge (x_3 \vee \overline{x_2}) \wedge (\overline{x_1} \vee \overline{x_4})$$

Example

Formula

$$\phi = (x_1 \vee \overline{x_2}) \wedge (x_2 \vee x_4) \wedge (\overline{x_1} \vee \overline{x_3}) \wedge (x_3 \vee \overline{x_2}) \wedge (\overline{x_1} \vee \overline{x_4})$$

Satisfying assignment:

$$x_1 = \bot \quad x_2 = \bot \quad x_3 = \top \quad x_4 = \top$$

Example

Formula

$$\phi = (x_1 \vee \overline{x_2}) \wedge (x_2 \vee x_4) \wedge (\overline{x_1} \vee \overline{x_3}) \wedge (x_3 \vee \overline{x_2}) \wedge (\overline{x_1} \vee \overline{x_4})$$

Satisfying assignment:

$$x_1 = \bot \quad x_2 = \bot \quad x_3 = \top \quad x_4 = \top$$

Trivial solution $O^*(2^n)$

An Observation

- A clause of the form $(\neg x_i \vee x_j)$ corresponds to $x_i \implies x_j$
- If $x_i = \top$ then, in any satisfying assignment, $x_j = \top$
- We say that x_j is **implied**.

- The same holds for any clause $C_j = (c_j^{(1)} \vee c_j^{(2)})$
- If $c_j^{(1)} = \bot$, then $c_j^{(2)} = \top$
- We say that the variable x_k corresponding to $c_j^{(2)}$ **implied**.
- If $c_j^{(2)} = x_k$, then $x_k = \top$. If $c_j^{(2)} = \overline{x_k}$, then $x_k = \bot$.

Example

$$\phi = (x_1 \vee \overline{x_2}) \wedge (x_2 \vee x_4) \wedge (\overline{x_1} \vee \overline{x_3}) \wedge (x_3 \vee \overline{x_2}) \wedge (\overline{x_1} \vee \overline{x_4})$$

Example

$$\phi = (x_1 \vee \overline{x_2}) \wedge (x_2 \vee x_4) \wedge (\overline{x_1} \vee \overline{x_3}) \wedge (x_3 \vee \overline{x_2}) \wedge (\overline{x_1} \vee \overline{x_4})$$

Assume that $x_1 = \top$

Example

$$\phi = (x_1 \vee \overline{x_2}) \wedge (x_2 \vee x_4) \wedge (\overline{x_1} \vee \underline{\overline{x_3}}) \wedge (x_3 \vee \overline{x_2}) \wedge (\overline{x_1} \vee \underline{\overline{x_4}})$$

Assume that $x_1 = \top$

x_3 and x_4 are implied. $x_3 = \bot$ and $x_4 = \bot$

Example

$$\phi = (x_1 \vee \overline{x_2}) \wedge (\underline{x_2} \vee x_4) \wedge (\overline{x_1} \vee \underline{\overline{x_3}}) \wedge (x_3 \vee \overline{x_2}) \wedge (\overline{x_1} \vee \underline{\overline{x_4}})$$

Assume that $x_1 = \top$

x_3 and x_4 are implied. $x_3 = \bot$ and $x_4 = \bot$

$x_2 = \top$ is implied

Example

$$\phi = (x_1 \vee \overline{x_2}) \wedge (\underline{x_2} \vee x_4) \wedge (\overline{x_1} \vee \underline{\overline{x_3}}) \wedge (x_3 \vee \underline{\overline{x_2}}) \wedge (\overline{x_1} \vee \underline{\overline{x_4}})$$

Assume that $x_1 = \top$

x_3 and x_4 are implied. $x_3 = \bot$ and $x_4 = \bot$

$x_2 = \top$ is implied

$x_2 = \bot$ is implied, a contradiction!

Example

$$\phi = (x_1 \vee \overline{x_2}) \wedge (x_2 \vee x_4) \wedge (\overline{x_1} \vee \overline{x_3}) \wedge (x_3 \vee \overline{x_2}) \wedge (\overline{x_1} \vee \overline{x_4})$$

Assume that $x_1 = \perp$

Example

$$\phi = (x_1 \vee \underline{\overline{x_2}}) \wedge (x_2 \vee x_4) \wedge (\overline{x_1} \vee \overline{x_3}) \wedge (x_3 \vee \overline{x_2}) \wedge (\overline{x_1} \vee \overline{x_4})$$

Assume that $x_1 = \perp$

$x_2 = \perp$ is implied.

Example

$$\phi = (x_1 \vee \underline{\overline{x_2}}) \wedge (x_2 \vee \underline{x_4}) \wedge (\overline{x_1} \vee \overline{x_3}) \wedge (x_3 \vee \overline{x_2}) \wedge (\overline{x_1} \vee \overline{x_4})$$

Assume that $x_1 = \bot$

$x_2 = \bot$ is implied.

$x_4 = \top$ is implied.

Example

$$\phi = (x_1 \vee \overline{x_2}) \wedge (x_2 \vee x_4) \wedge (\overline{x_1} \vee \overline{x_3}) \wedge (x_3 \vee \overline{x_2}) \wedge (\overline{x_1} \vee \overline{x_4})$$

Assume that $x_1 = \bot$

$x_2 = \bot$ is implied.

$x_4 = \top$ is implied.

We found a satisfying assignment.

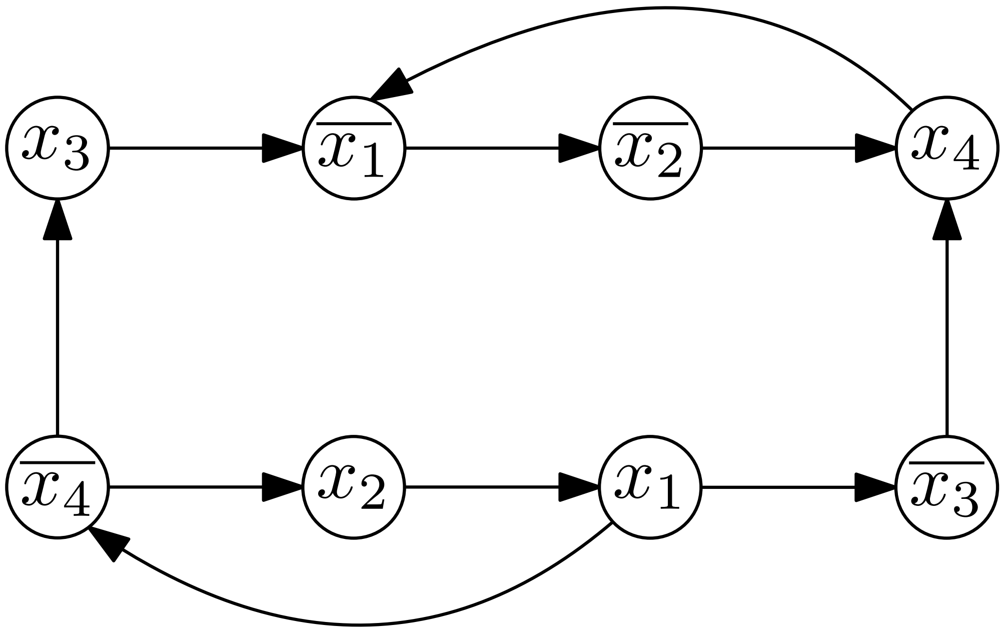
The Implication Graph

Given ϕ we construct a directed graph $G_\phi = (V, E)$ where:

- The vertices of G are all possible literals of ϕ, i.e., for each variable x_i we add both x_i and $\overline{x_i}$ to V.
- For each clause $(\ell_i \vee \ell_j)$:
  - Add $(\overline{\ell_i}, \ell_j)$ to E
  - Add $(\overline{\ell_j}, \ell_i)$ to E
- Intuitively $(u, v) \in E$ means that if $u = \top$, then we must set $v = \top$.

Example

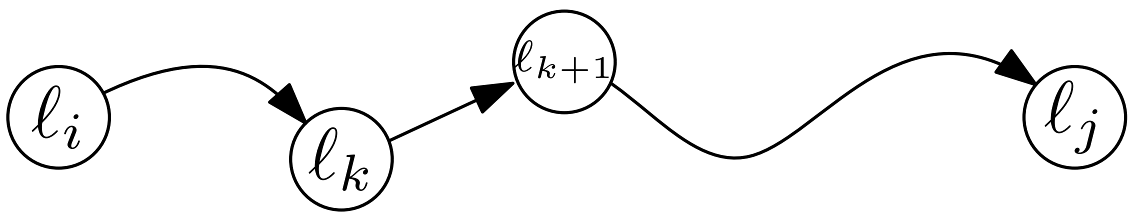
$$\phi = (x_1 \vee \overline{x_2}) \wedge (x_2 \vee x_4) \wedge (\overline{x_1} \vee \overline{x_3}) \wedge (x_3 \vee x_4) \wedge (\overline{x_1} \vee \overline{x_4})$$



A useful property

Claim: G_ϕ is skew-simmetric: If there is a path P from ℓ_i to ℓ_j in G_ϕ, then there is also a path from $\overline{\ell_j}$ to $\overline{\ell_i}$.

- Pick any edge (ℓ_k, ℓ_{k+1}) of P.



- The edge (ℓ_k, ℓ_{k+1}) must have been created from the clause $(\overline{\ell_k} \vee \ell_{k+1})$.

- The clause $(\overline{\ell_k} \vee \ell_{k+1})$ also creates the edge $(\overline{\ell_{k+1}}, \overline{\ell_k})$.

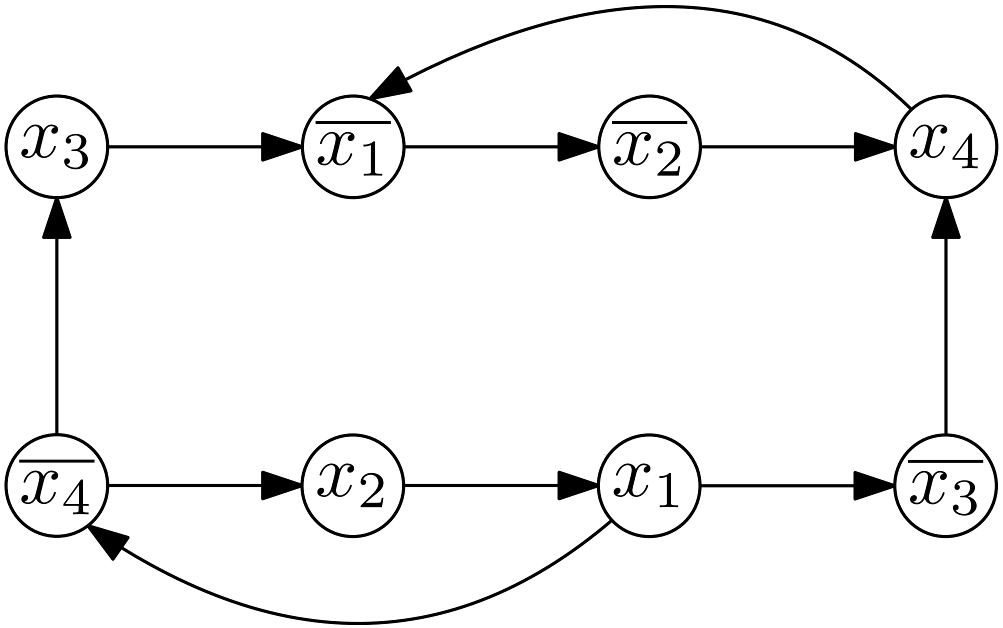
Strongly Connected Components

Definition: A strongly connected component of a graph $G = (V, E)$ is a maximal set $C \subseteq V$ such that $\forall x, y \in C$, there is a path from x to y in G.

Strongly Connected Components

Definition: A strongly connected component of a graph $G = (V, E)$ is a maximal set $C \subseteq V$ such that $\forall x, y \in C$, there is a path from x to y in G.

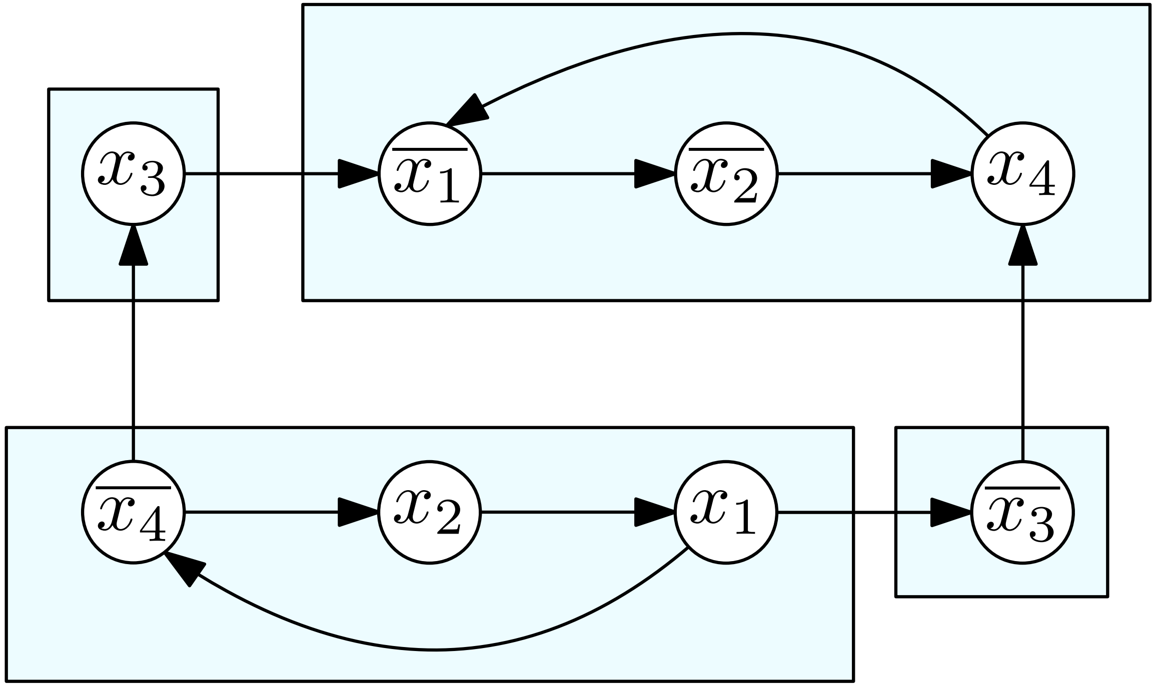
$$\phi = (x_1 \vee \overline{x_2}) \wedge (x_2 \vee x_4) \wedge (\overline{x_1} \vee \overline{x_3}) \wedge (x_3 \vee x_4) \wedge (\overline{x_1} \vee \overline{x_4})$$



Strongly Connected Components

Definition: A strongly connected component of a graph $G = (V, E)$ is a maximal set $C \subseteq V$ such that $\forall x, y \in C$, there is a path from x to y in G.

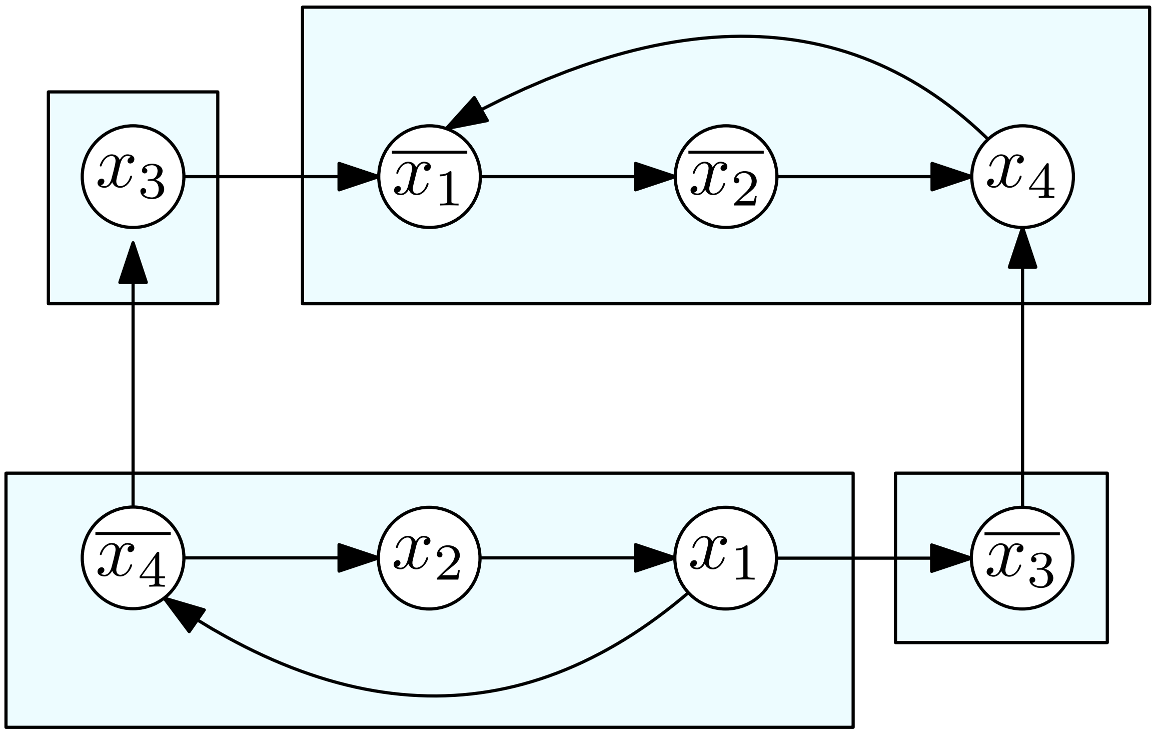
$$\phi = (x_1 \vee \overline{x_2}) \wedge (x_2 \vee x_4) \wedge (\overline{x_1} \vee \overline{x_3}) \wedge (x_3 \vee x_4) \wedge (\overline{x_1} \vee \overline{x_4})$$



Contracted Graph

Construct a new graph $G'_\phi = (V', E')$ from $G_\phi = (V, E)$:

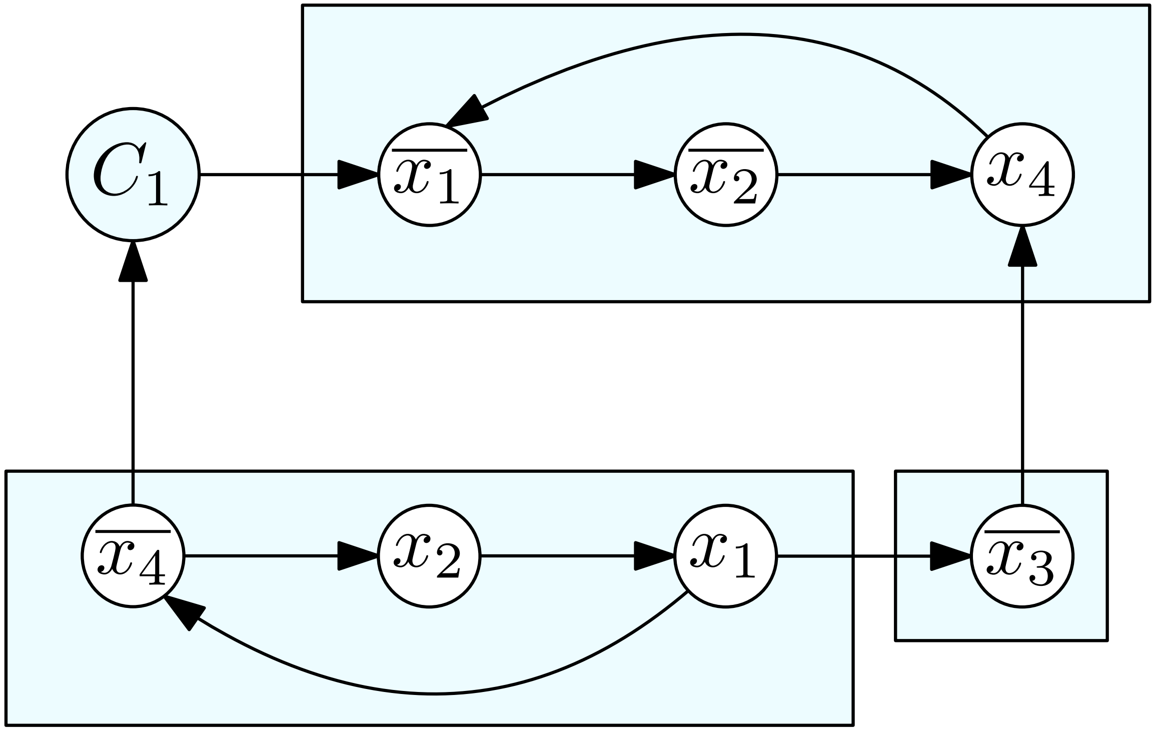
- Each vertex in V' is a SCC of G.
- There is an edge between a pair of distinct connected components $(C, C') \in E$ iff $\exists x \in C, y \in C'$ such that $(x, y) \in E$.



Contracted Graph

Construct a new graph $G'_\phi = (V', E')$ from $G_\phi = (V, E)$:

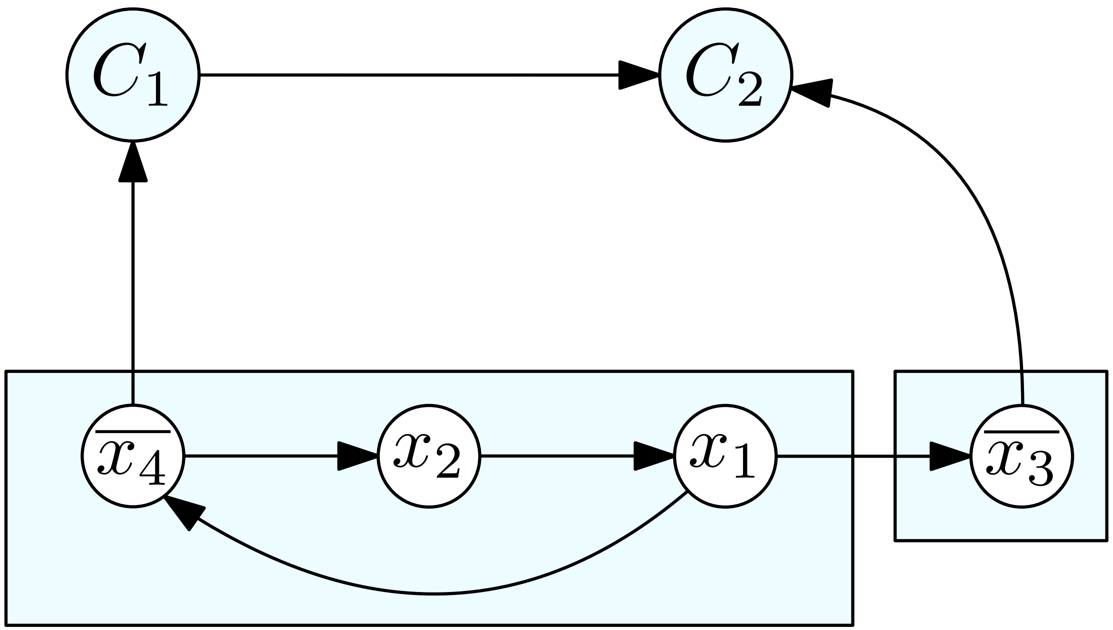
- Each vertex in V' is a SCC of G.
- There is an edge between a pair of distinct connected components $(C, C') \in E$ iff $\exists x \in C, y \in C'$ such that $(x, y) \in E$.



Contracted Graph

Construct a new graph $G'_\phi = (V', E')$ from $G_\phi = (V, E)$:

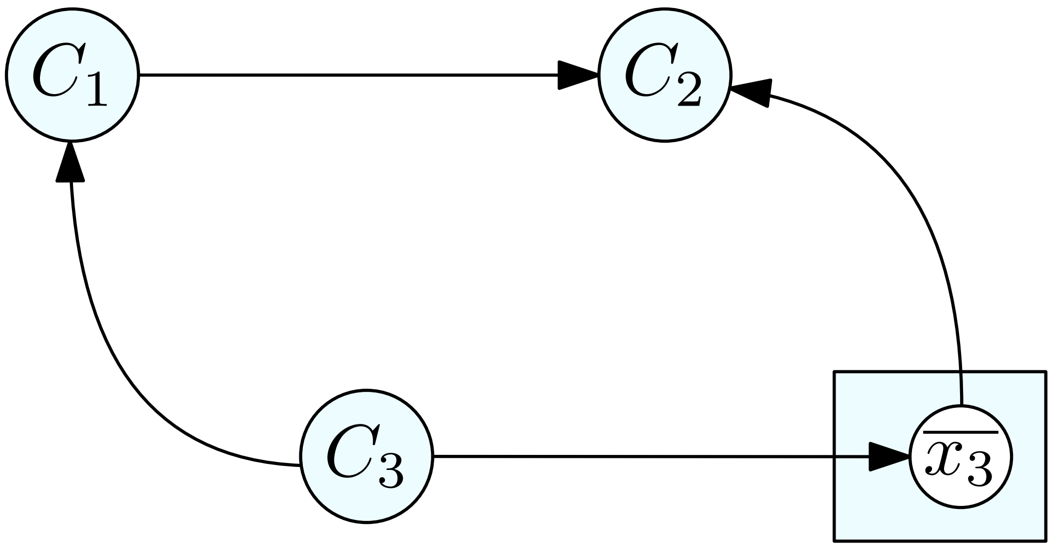
- Each vertex in V' is a SCC of G.
- There is an edge between a pair of distinct connected components $(C, C') \in E$ iff $\exists x \in C, y \in C'$ such that $(x, y) \in E$.



Contracted Graph

Construct a new graph $G'_\phi = (V', E')$ from $G_\phi = (V, E)$:

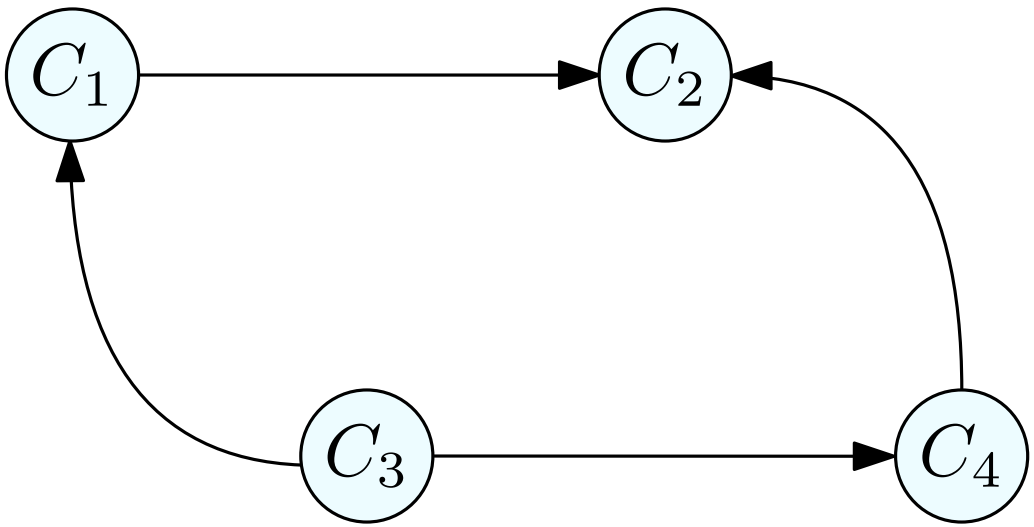
- Each vertex in V' is a SCC of G.
- There is an edge between a pair of distinct connected components $(C, C') \in E$ iff $\exists x \in C, y \in C'$ such that $(x, y) \in E$.



Contracted Graph

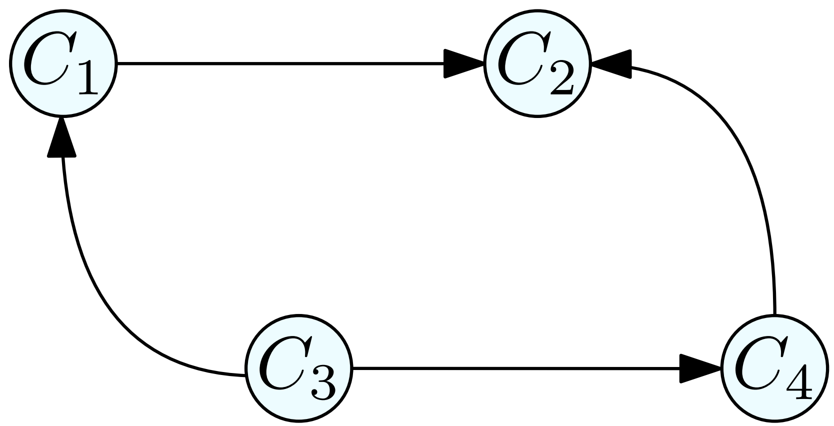
Construct a new graph $G'_\phi = (V', E')$ from $G_\phi = (V, E)$:

- Each vertex in V' is a SCC of G.
- There is an edge between a pair of distinct connected components $(C, C') \in E$ iff $\exists x \in C, y \in C'$ such that $(x, y) \in E$.



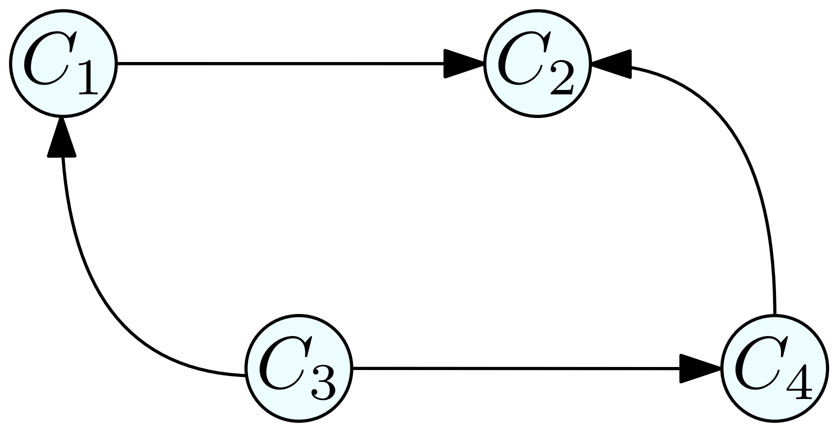
Toplogical Order

Observation: Contracting the SCCs of a directed graph yields a directed acyclic graph.

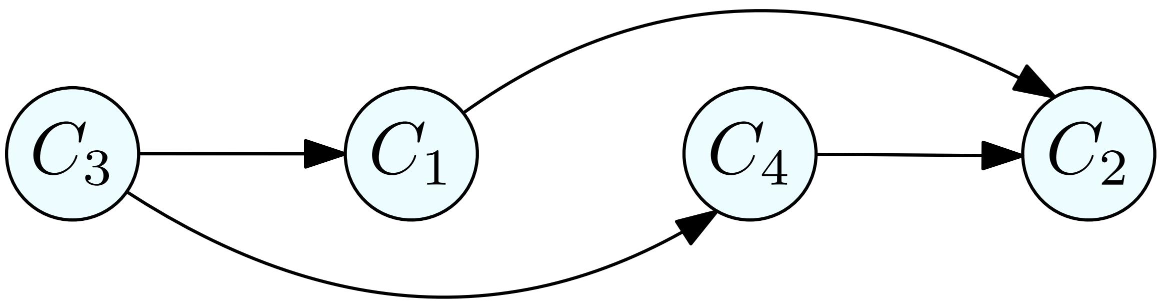


Toplogical Order

Observation: Contracting the SCCs of a directed graph yields a directed acyclic graph.



Definition: A topological order of a directed acyclic graph is a linear order $v_1, v_2, \ldots$ of the vertices such that, for any edge (v_i, v_j), we have $i < j$.



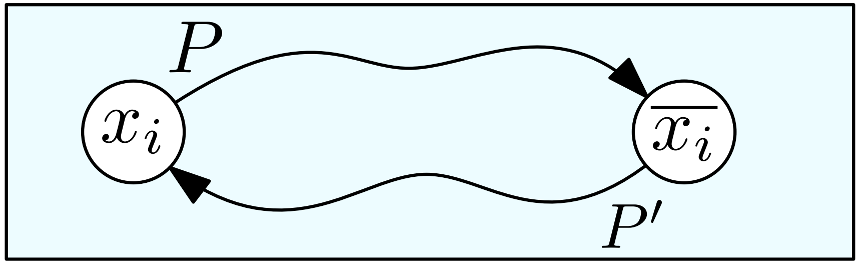
Relation between SSCs and 2-SAT

Claim 1: If, for some x_i, both x_i and $\overline{x_i}$ belong to the same SCC C, then ϕ is not satisfiable.

Relation between SSCs and 2-SAT

Claim 1: If, for some x_i, both x_i and $\overline{x_i}$ belong to the same SCC C, then ϕ is not satisfiable.

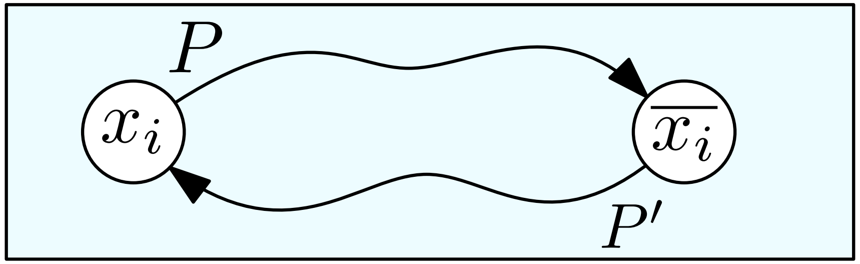
- Since x_i and $\overline{x_i}$ are in the same SCC, there is a path P in G from x_i to $\overline{x_i}$ and a path P' from $\overline{x_i}$ to x_i.



Relation between SSCs and 2-SAT

Claim 1: If, for some x_i, both x_i and $\overline{x_i}$ belong to the same SCC C, then ϕ is not satisfiable.

- Since x_i and $\overline{x_i}$ are in the same SCC, there is a path P in G from x_i to $\overline{x_i}$ and a path P' from $\overline{x_i}$ to x_i.

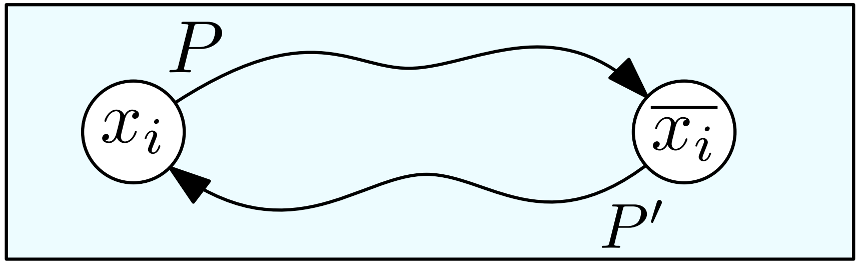


- In any satisfying assignment, we cannot have $x_i = \top$, since it would imply (through P) that $\overline{x_i} = \top$, i.e., $x_i = \bot$. ↯

Relation between SSCs and 2-SAT

Claim 1: If, for some x_i, both x_i and $\overline{x_i}$ belong to the same SCC C, then ϕ is not satisfiable.

- Since x_i and $\overline{x_i}$ are in the same SCC, there is a path P in G from x_i to $\overline{x_i}$ and a path P' from $\overline{x_i}$ to x_i.



- In any satisfying assignment, we cannot have $x_i = \top$, since it would imply (through P) that $\overline{x_i} = \top$, i.e., $x_i = \bot$. ↯
- A symmetric argument shows that we cannot have $x_i = \bot$ since it would imply $x_i = \top$ through P'. ↯

Relation between SSCs and 2-SAT

Assumption: for all x_i, x_i and $\overline{x_i}$ belong to different SCCs.

An algorithm:

- $\forall$ SCC $C = C_1, C_2, \ldots$ of G in reverse topological order.
  - Assign all unassigned literals of C to $\top$ and their complement to $\bot$.

Relation between SSCs and 2-SAT

Assumption: for all x_i, x_i and $\overline{x_i}$ belong to different SCCs.

An algorithm:

- $\forall$ SCC $C = C_1, C_2, \dots$ of G in reverse topological order.
  - Assign all unassigned literals of C to $\top$ and their complement to $\bot$.

Claim 2: When ℓ_i is set to $\top$, all literals ℓ_j reachable from ℓ_i in G are set to $\top$.

Proof: By induction on the index k of the SCC C_k containing ℓ_i.

Relation between SSCs and 2-SAT

Suppose that there is a neighbor ℓ_j of ℓ_i such that $\ell_j = \bot$.

By skew-simmetry G contains the edge $(\overline{\ell_j}, \overline{\ell_i})$

$\overline{\ell_j}$ is set to $\top$ and must belong to a SCC C_h for some $h \leq k$.

If $h = k$:

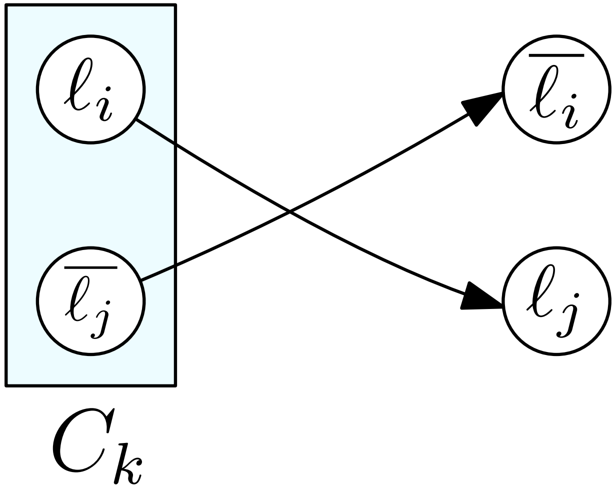
Relation between SSCs and 2-SAT

Suppose that there is a neighbor ℓ_j of ℓ_i such that $\ell_j = \bot$.

By skew-simmetry G contains the edge $(\overline{\ell_j}, \overline{\ell_i})$

$\overline{\ell_j}$ is set to $\top$ and must belong to a SCC C_h for some $h \leq k$.

If $h = k$:



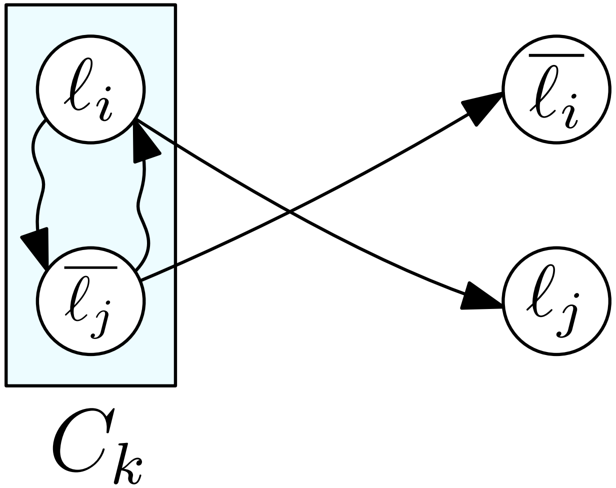
Relation between SSCs and 2-SAT

Suppose that there is a neighbor ℓ_j of ℓ_i such that $\ell_j = \bot$.

By skew-simmetry G contains the edge $(\overline{\ell_j}, \overline{\ell_i})$

$\overline{\ell_j}$ is set to $\top$ and must belong to a SCC C_h for some $h \leq k$.

If $h = k$:



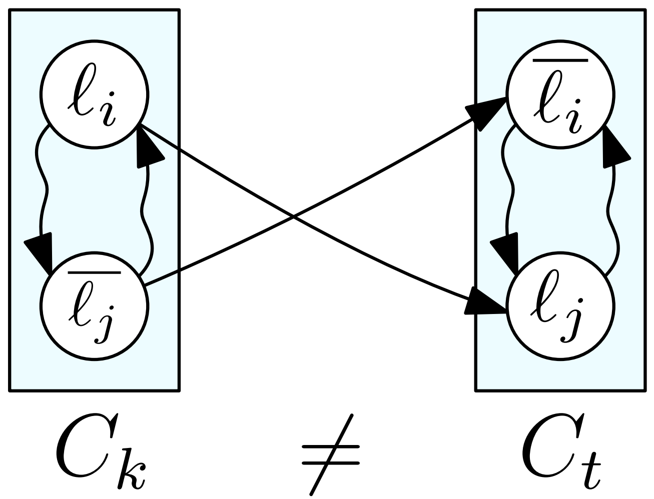
Relation between SSCs and 2-SAT

Suppose that there is a neighbor ℓ_j of ℓ_i such that $\ell_j = \bot$.

By skew-simmetry G contains the edge $(\overline{\ell_j}, \overline{\ell_i})$

$\overline{\ell_j}$ is set to $\top$ and must belong to a SCC C_h for some $h \leq k$.

If $h = k$:



- $C_k \neq C_t$ (otherwise $\ell_i, \overline{\ell_i} \in C_k$)

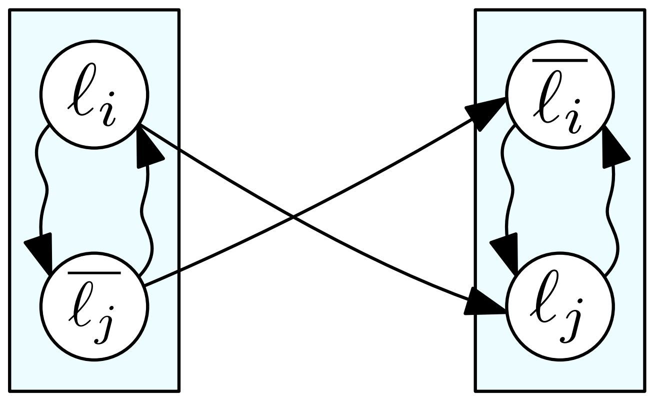
Relation between SSCs and 2-SAT

Suppose that there is a neighbor ℓ_j of ℓ_i such that $\ell_j = \bot$.

By skew-simmetry G contains the edge $(\overline{\ell_j}, \overline{\ell_i})$

$\overline{\ell_j}$ is set to $\top$ and must belong to a SCC C_h for some $h \leq k$.

If $h = k$:



$C_k \neq C_t$

- $C_k \neq C_t$ (otherwise $\ell_i, \overline{\ell_i} \in C_k$)
- $(C_k, C_t) \in E' \implies t < k$

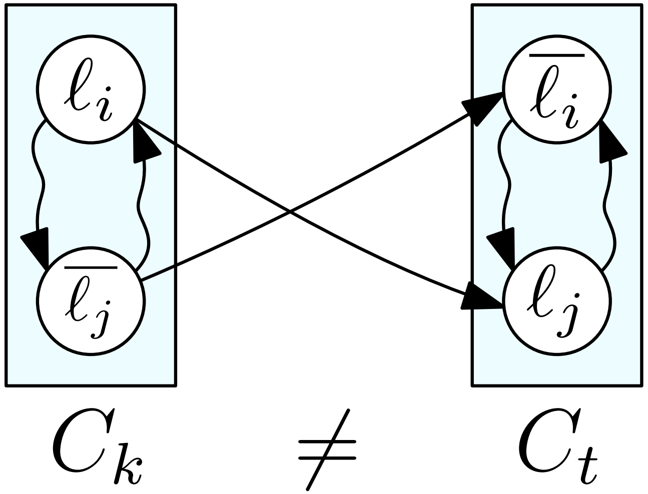
Relation between SSCs and 2-SAT

Suppose that there is a neighbor ℓ_j of ℓ_i such that $\ell_j = \bot$.

By skew-simmetry G contains the edge $(\overline{\ell_j}, \overline{\ell_i})$

$\overline{\ell_j}$ is set to $\top$ and must belong to a SCC C_h for some $h \leq k$.

If $h = k$:



- $C_k \neq C_t$ (otherwise $\ell_i, \overline{\ell_i} \in C_k$)
- $(C_k, C_t) \in E' \implies t < k$
- After C_t was considered
  $\overline{\ell_i} = \top \implies \ell_i = \bot$.

↯

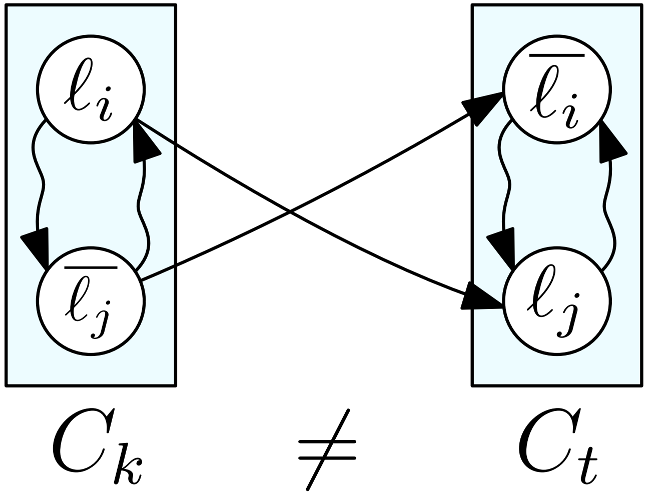
Relation between SSCs and 2-SAT

Suppose that there is a neighbor ℓ_j of ℓ_i such that $\ell_j = \bot$.

By skew-simmetry G contains the edge $(\overline{\ell_j}, \overline{\ell_i})$

$\overline{\ell_j}$ is set to $\top$ and must belong to a SCC C_h for some $h \leq k$.

If $h = k$:



- $C_k \neq C_t$ (otherwise $\ell_i, \overline{\ell_i} \in C_k$)
- $(C_k, C_t) \in E' \implies t < k$
- After C_t was considered $\overline{\ell_i} = \top \implies \ell_i = \bot$. ϟ

If $h < k$:

By inductive hyphotesis, all neighbors of $\overline{\ell_j}$ are set to $\top$, i.e., $\overline{\ell_i} = \top \implies \ell_i = \bot$. ϟ

Relation between SSCs and 2-SAT

Claim 1: If, for some x_i, both x_i and $\overline{x_i}$ belong to the same SCC C, then ϕ is not satisfiable.

Assumption: $\forall x_i$, x_i and $\overline{x_i}$ belong to different SCCs.

Claim 2: When ℓ_i is set to $\top$, all literals ℓ_j reachable from ℓ_i in G are set to $\top$.

Relation between SSCs and 2-SAT

Claim 1: If, for some x_i, both x_i and $\overline{x_i}$ belong to the same SCC C, then ϕ is not satisfiable.

Assumption: $\forall x_i$, x_i and $\overline{x_i}$ belong to different SCCs.

Claim 2: When ℓ_i is set to $\top$, all literals ℓ_j reachable from ℓ_i in G are set to $\top$.

Corollary: ϕ is satisfiable iff $\forall x_i$, x_i and $\overline{x_i}$ belong to different SCCs. The algorithm computes a satisfying assignment.

- Consider a generic clause $(\ell_i \vee \ell_j)$
- If ℓ_i is set to $\top$, the clause is satisfied.
- If ℓ_i is set to $\bot$: $\overline{\ell_i} = \top$ and G contains the edge $(\overline{\ell_i}, \ell_j)$. The claim implies that $\ell_j = \top$.

Time Complexity

Satisfiability

(Assuming $m = \Omega(n)$)

- Construct the implication graph G_ϕ — $O(m)$
- Compute the SSCs of G_ϕ — $O(m)$
- If a SCC of G contains both x_i and $\overline{x_i}$, for some x_i: — $O(n)$
  - Return "ϕ is not satisfiable"
- Return "ϕ is satisfiable"

Time Complexity

Satisfying assignment

(Assuming $m = \Omega(n)$)

- Construct the implication graph G_ϕ — $O(m)$
- Compute the SSCs of G_ϕ — $O(m)$
- If a SCC of G contains both x_i and $\overline{x_i}$, for some x_i: — $O(n)$
  - Return "ϕ is not satisfiable"
- $G'_\phi \leftarrow$ Contract the SCCs of G_ϕ — $O(m)$
- Topologically sort G' — $O(m)$
- $\forall$ SCC C of G in reverse topological order.
  - Assign all unassigned literals of C to $\top$ and their complement to $\bot$.

(last two lines together: $O(n)$)

Time Complexity

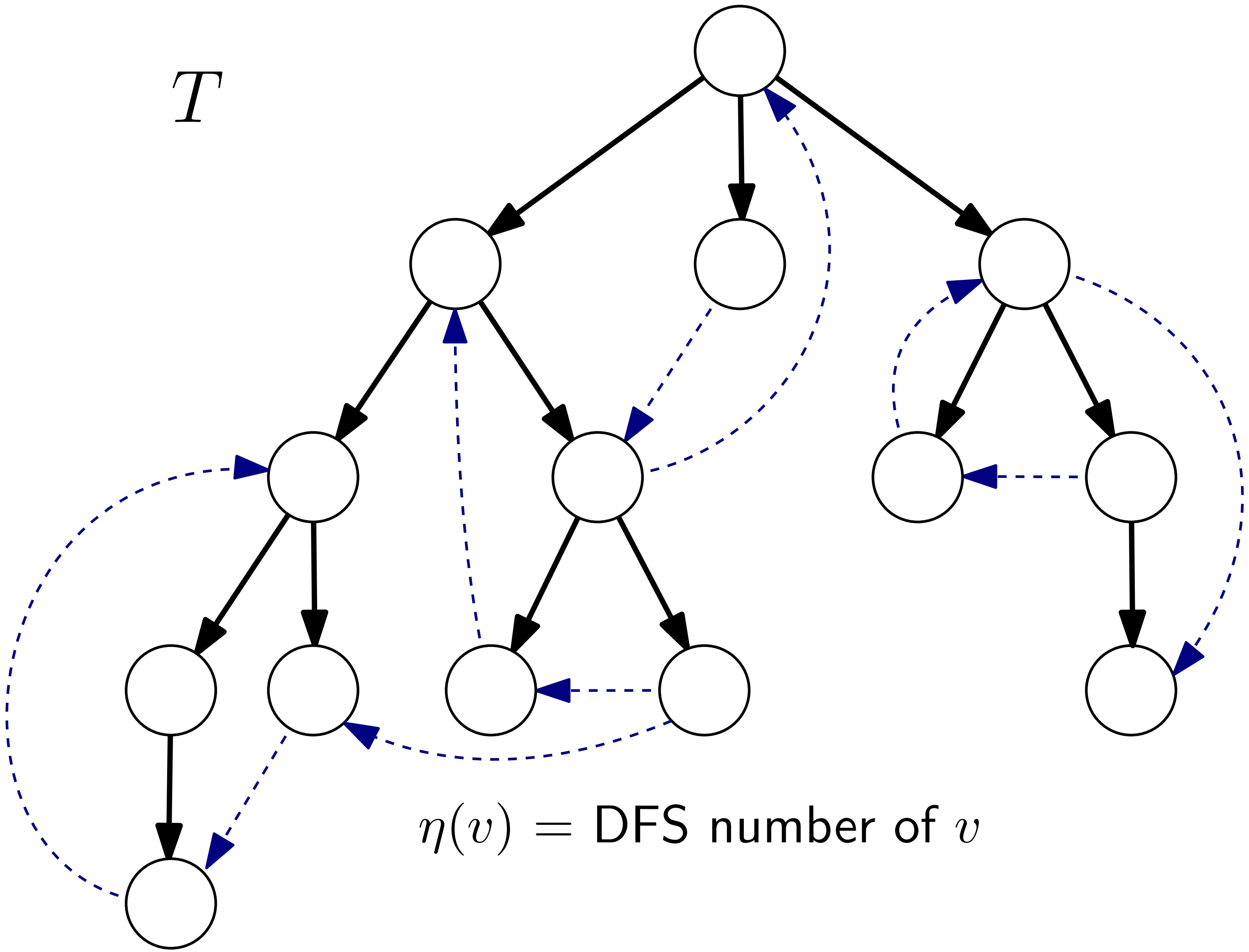
Satisfying assignment

(Assuming $m = \Omega(n)$)

- Construct the implication graph G_ϕ — $O(m)$
- Compute the SSCs of G_ϕ — How? — $O(m)$
- If a SCC of G contains both x_i and $\overline{x_i}$, for some x_i: — $O(n)$
  - Return "ϕ is not satisfiable"
- $G'_\phi \leftarrow$ Contract the SCCs of G_ϕ — $O(m)$
- Topologically sort G' — $O(m)$
- $\forall$ SCC C of G in reverse topological order.
  - Assign all unassigned literals of C to $\top$ and their complement to $\bot$.

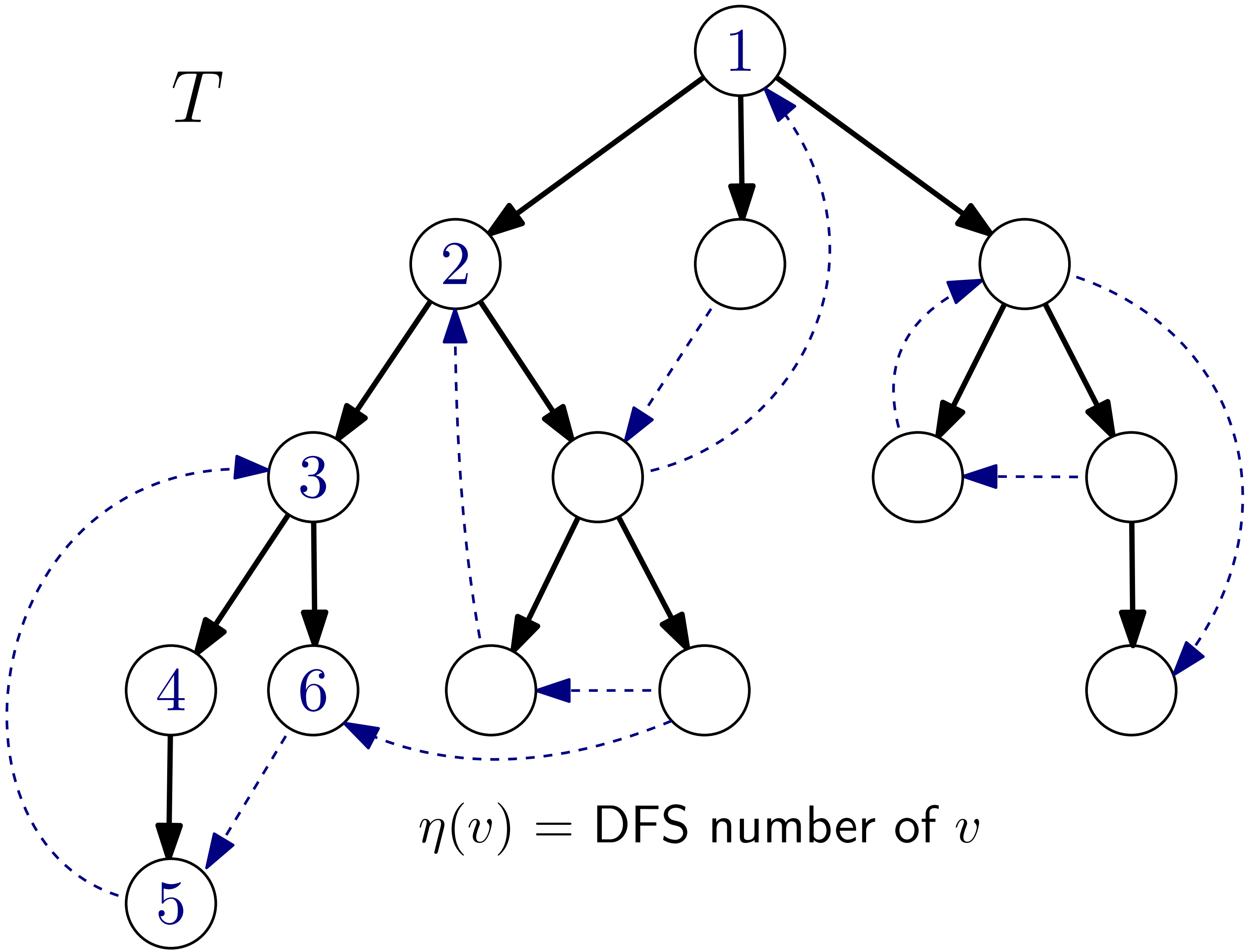
(last two steps together: $O(n)$)

Tarjan's algorithm



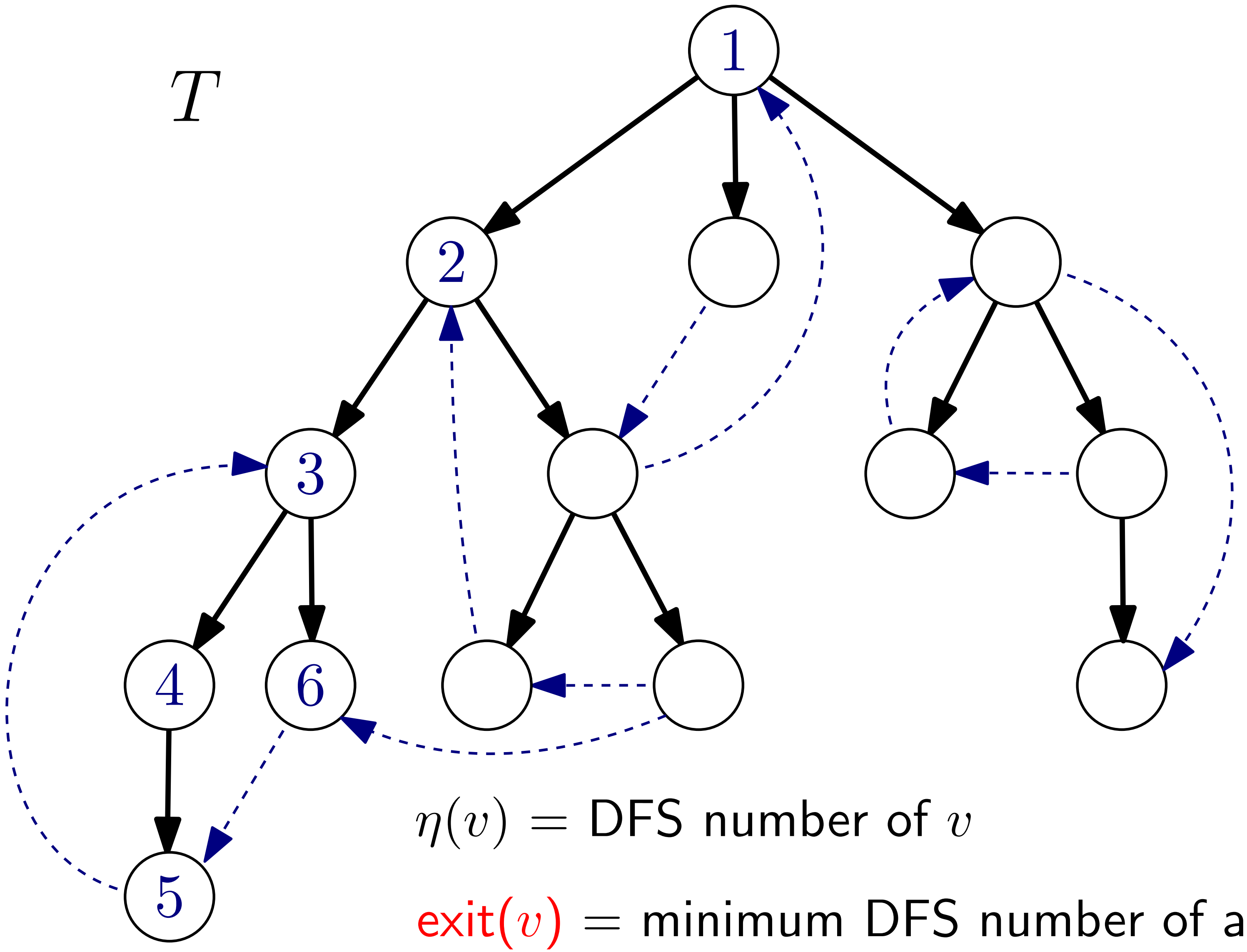
$\eta(v)$ = DFS number of v

Tarjan's algorithm



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

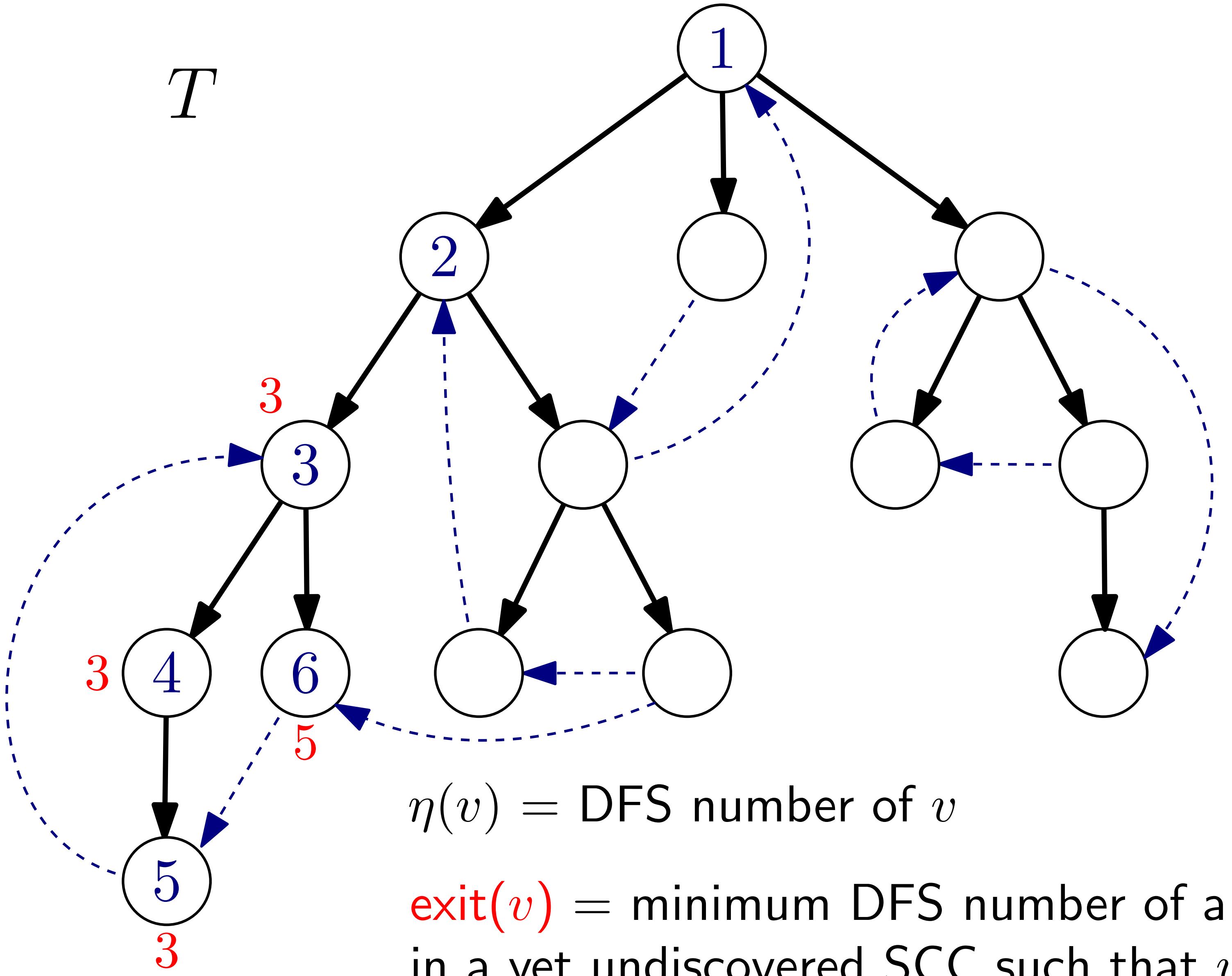
Tarjan's algorithm



$\eta(v)$ = 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.

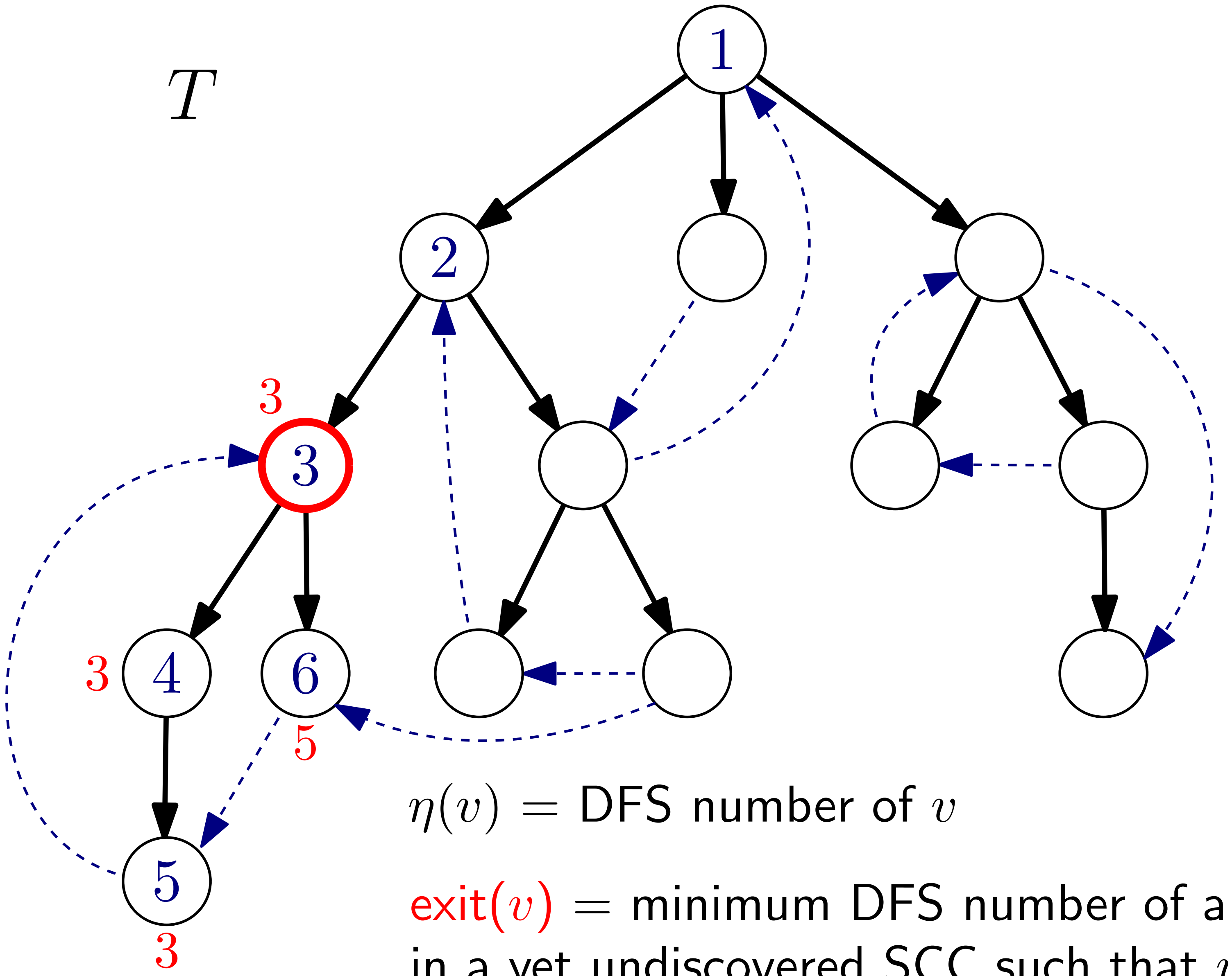
Tarjan's algorithm



$\eta(v)$ = 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.

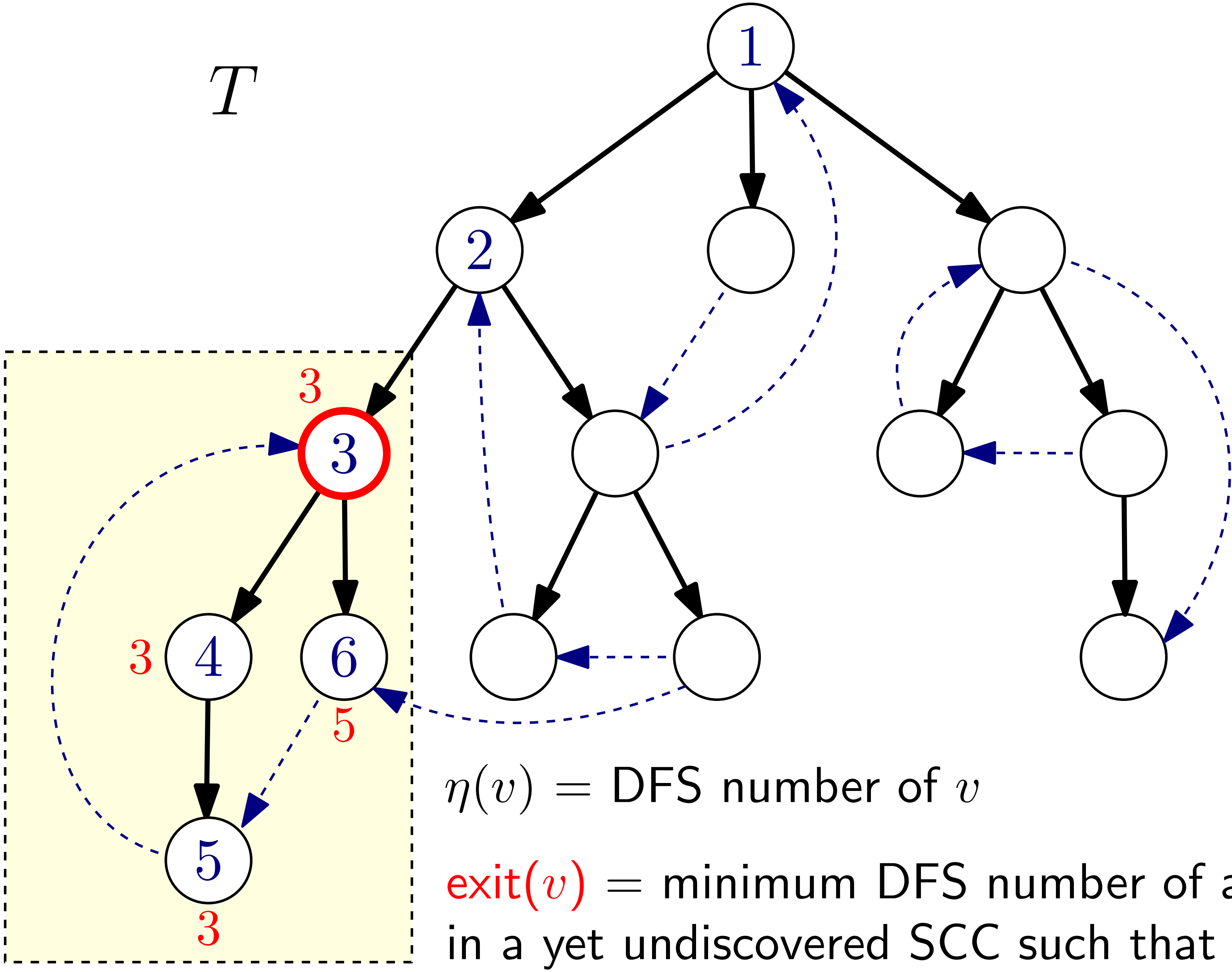
Tarjan's algorithm



$\eta(v)$ = 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.

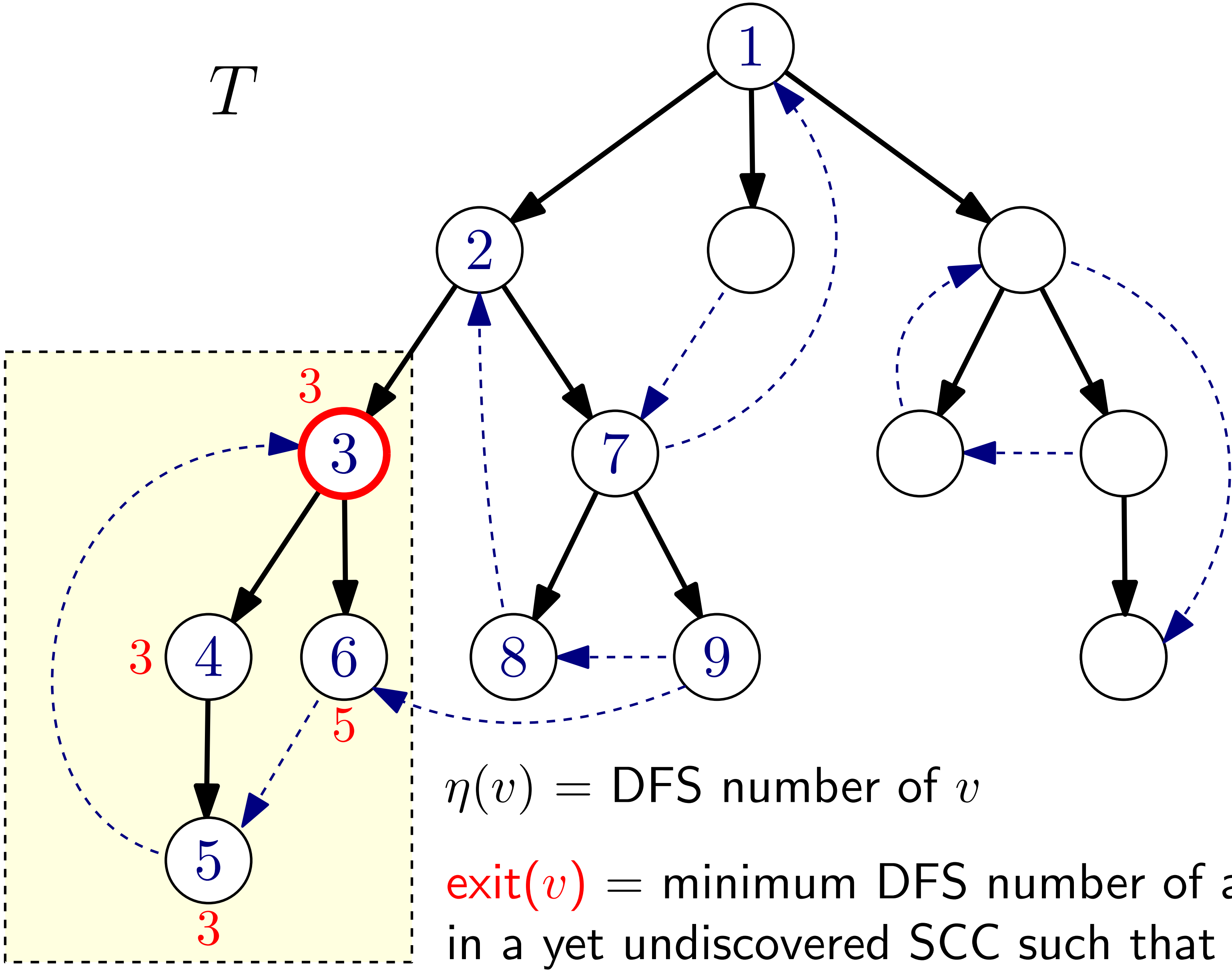
Tarjan's algorithm



$\eta(v)$ = 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.

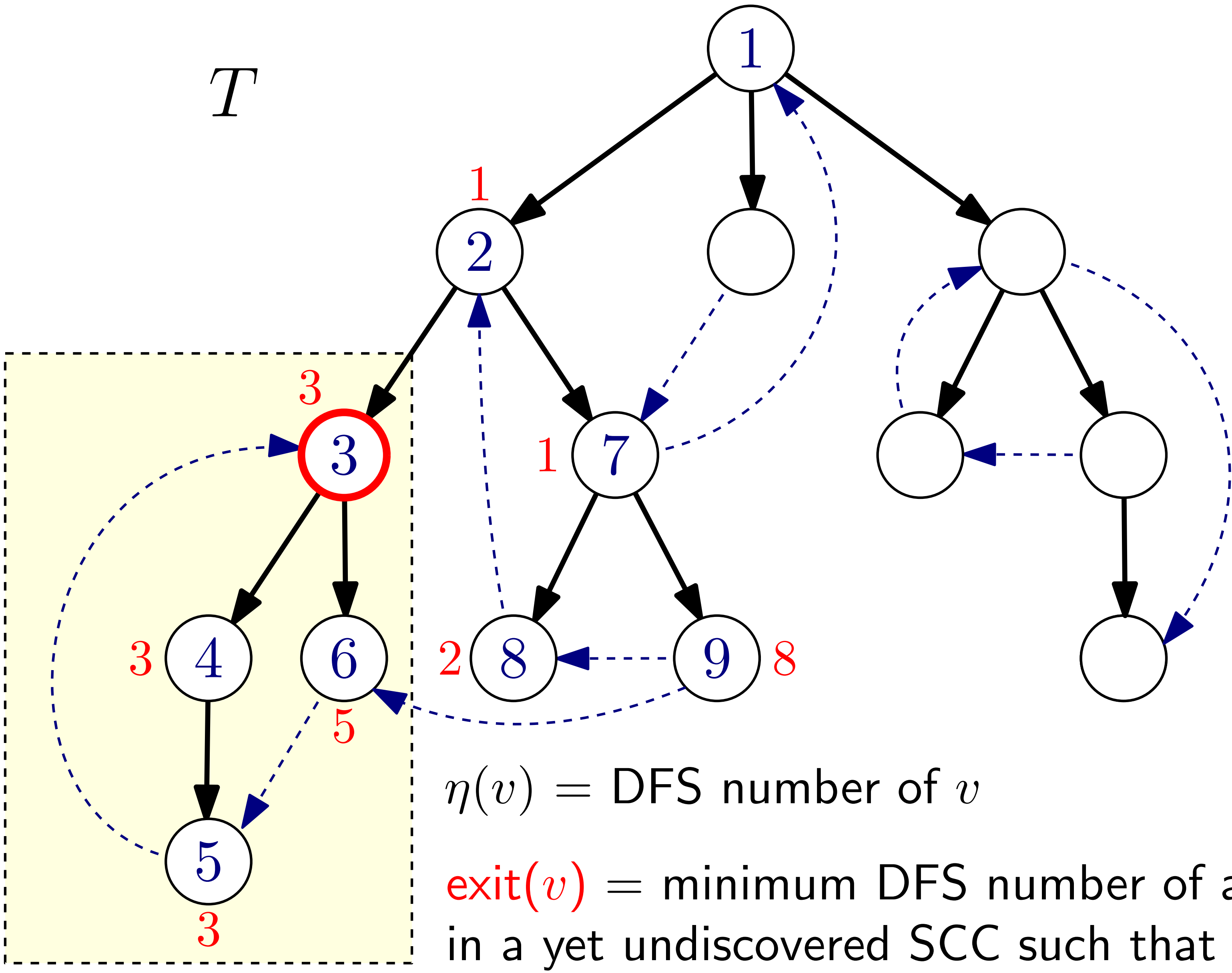
Tarjan's algorithm



$\eta(v)$ = 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.

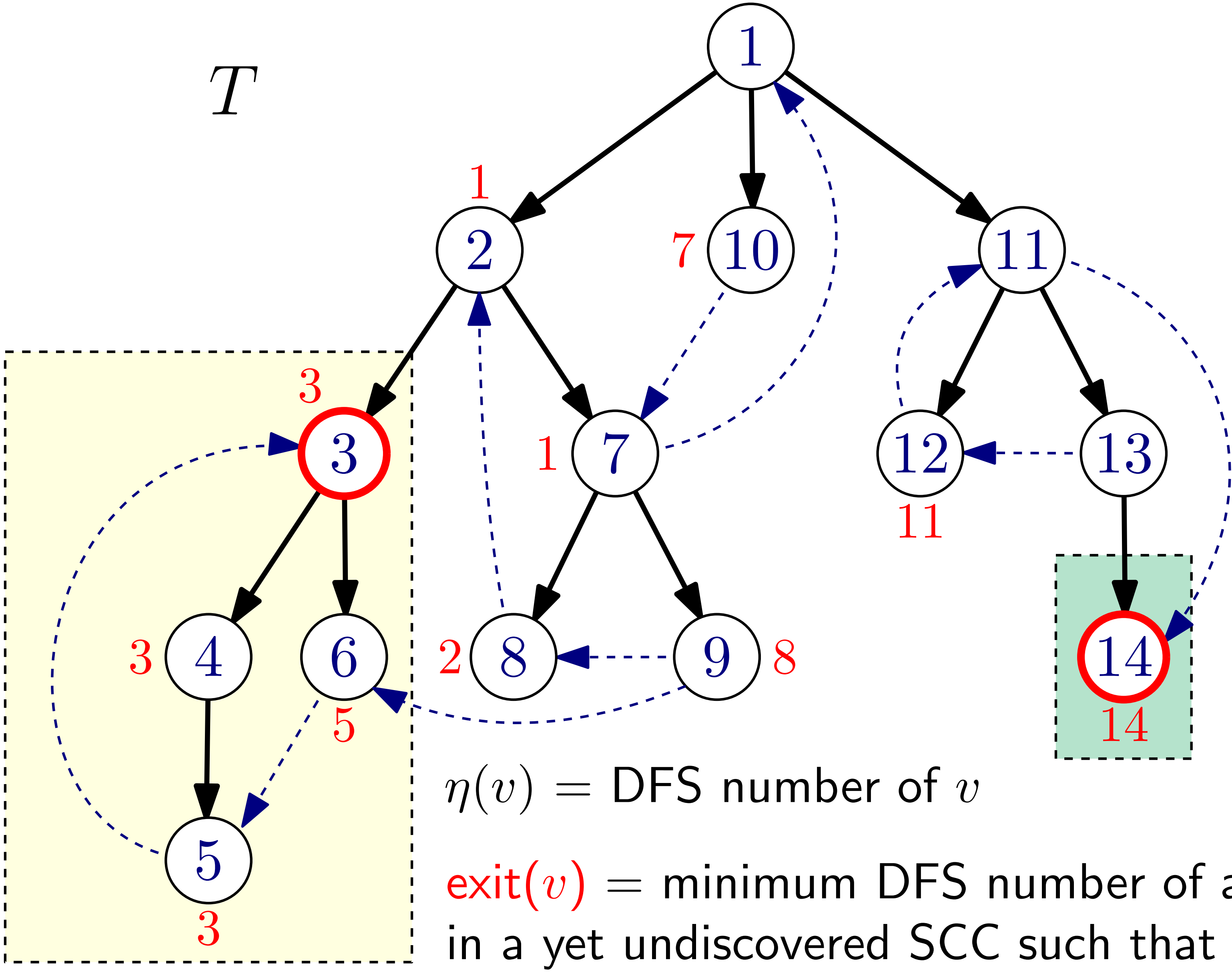
Tarjan's algorithm



$\eta(v)$ = 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.

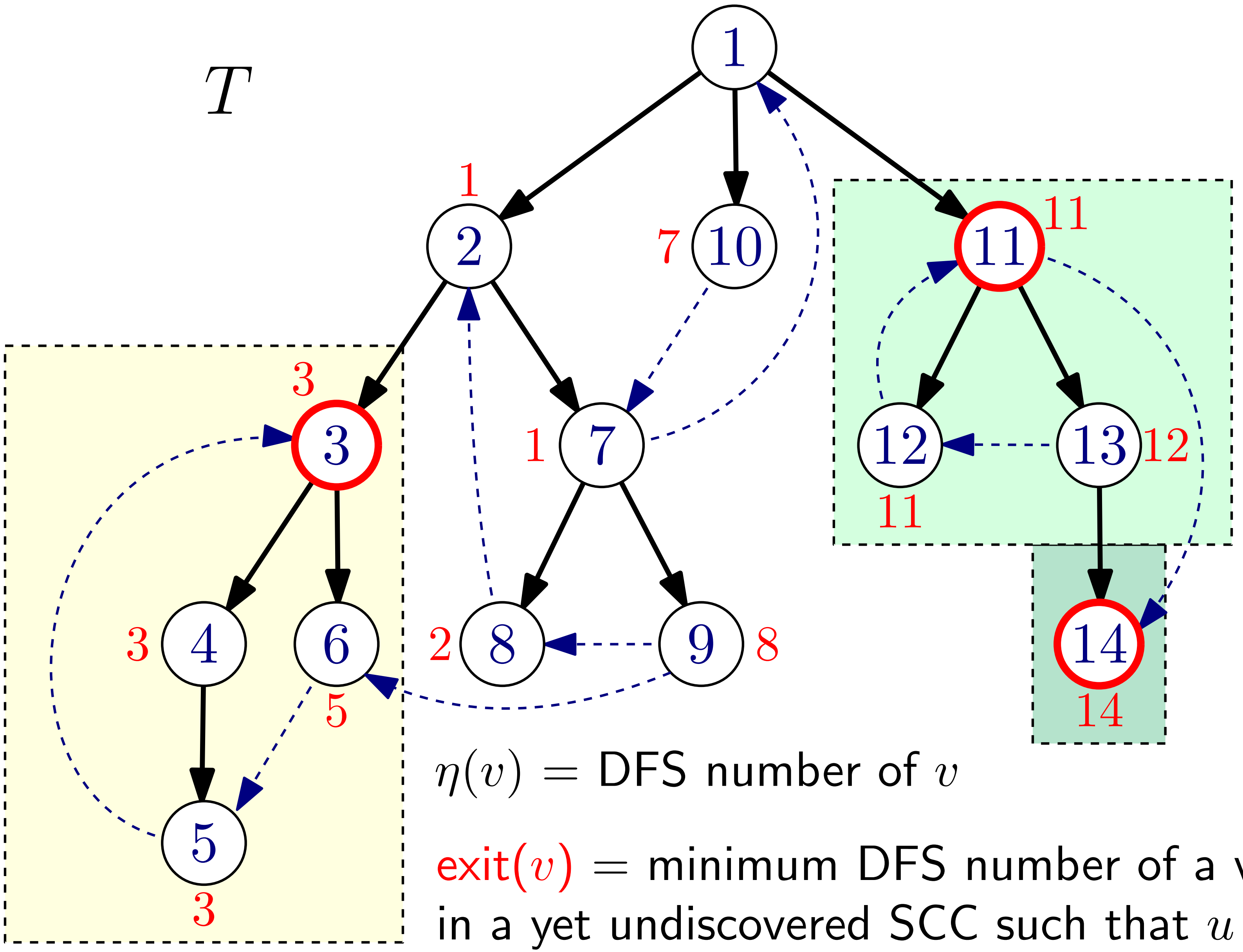
Tarjan's algorithm



$\eta(v)$ = 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.

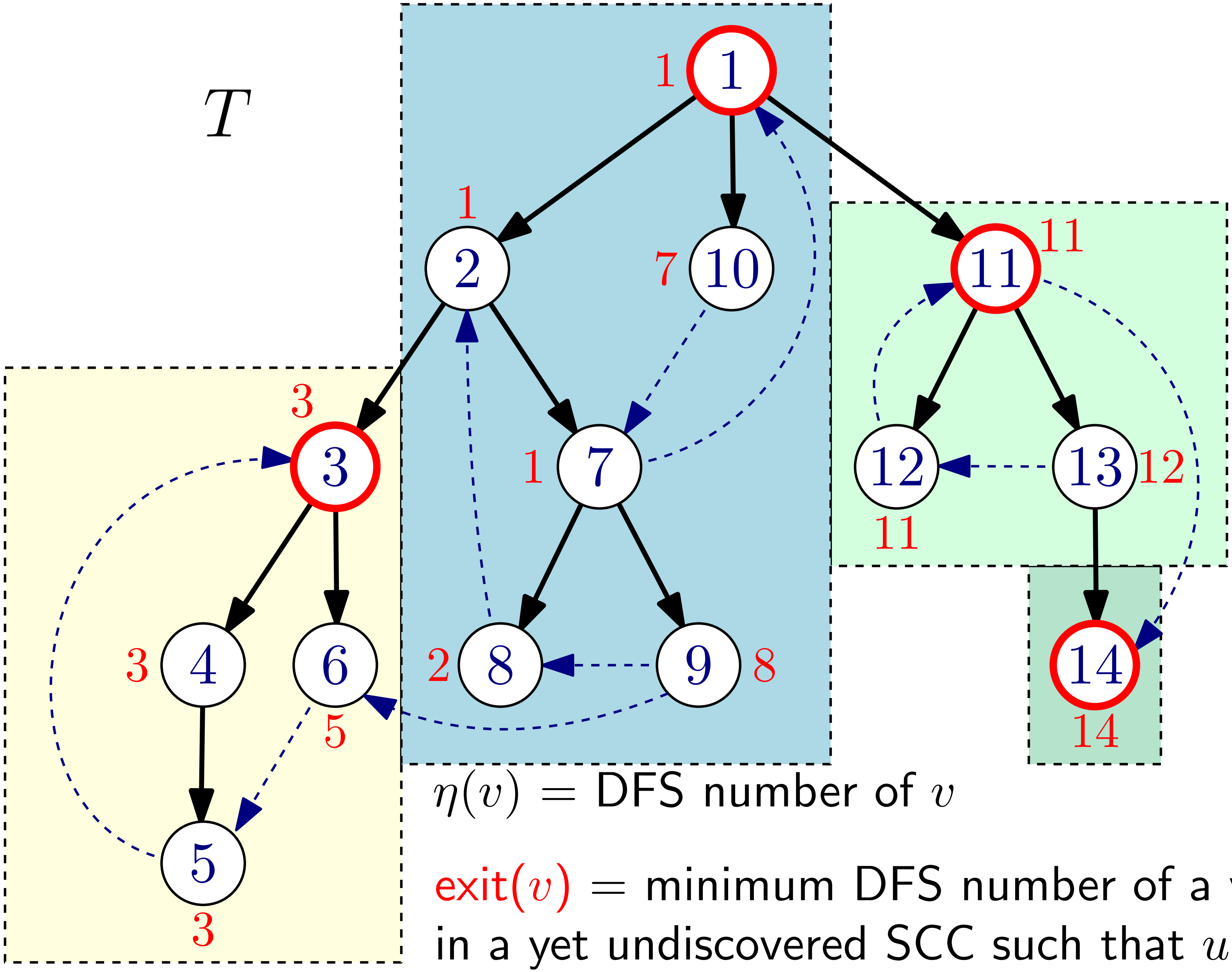
Tarjan's algorithm



$\eta(v)$ = 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

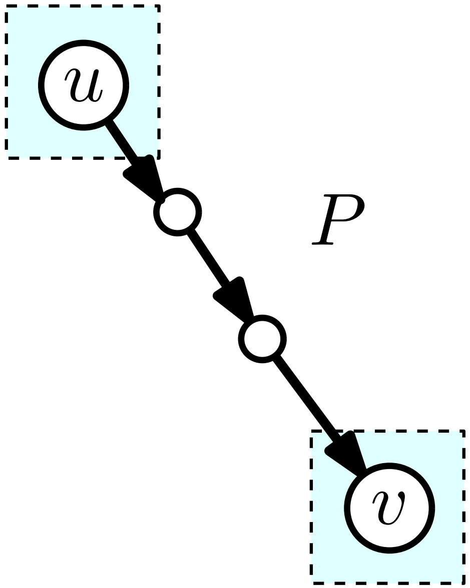
Claim: Let C be a SCC. The subgraph $T[C]$ of T induced by C is connected.

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

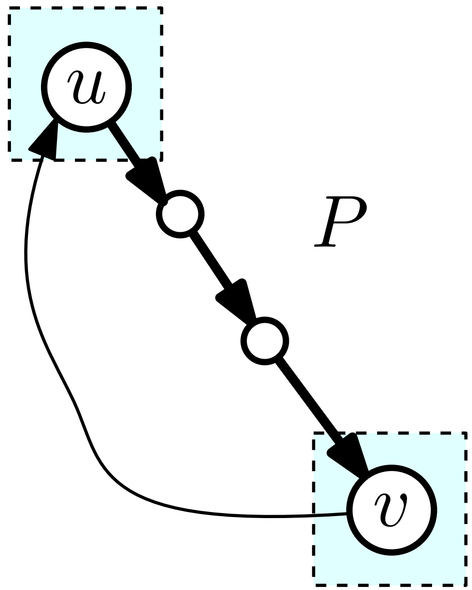
Claim: Let C be a SCC. The subgraph $T[C]$ of T induced by C is connected.

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

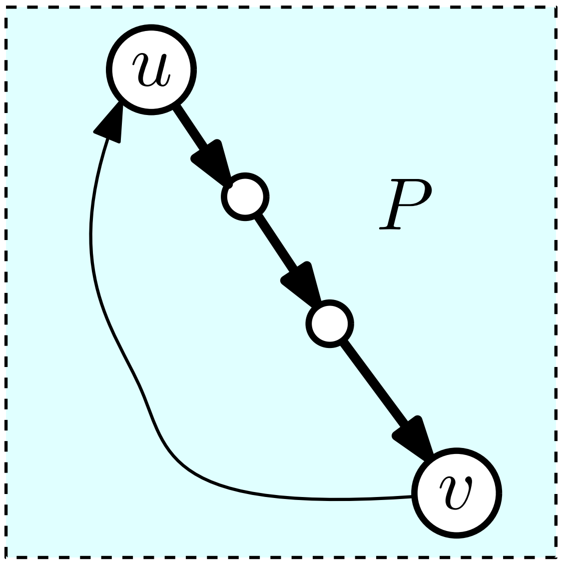
Claim: Let C be a SCC. The subgraph $T[C]$ of T induced by C is connected.

Proof:

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

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

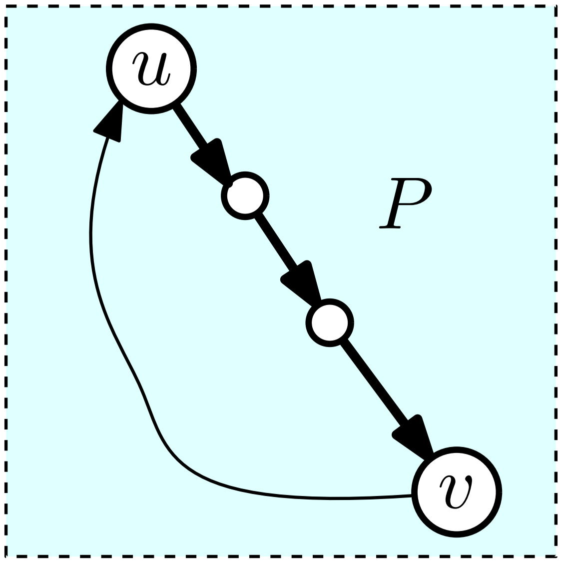
- u must be an ancestor of v in T (by the properties of DFS).



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 $T[C]$.

$\square$

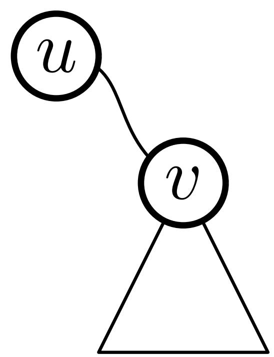
Proof of correctness

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

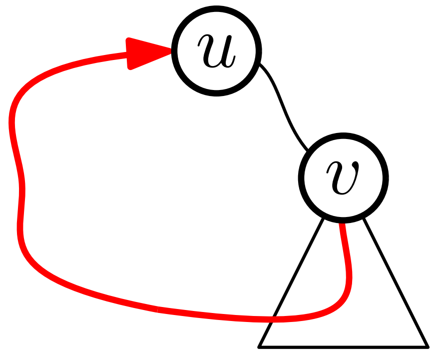


Proof of correctness

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.

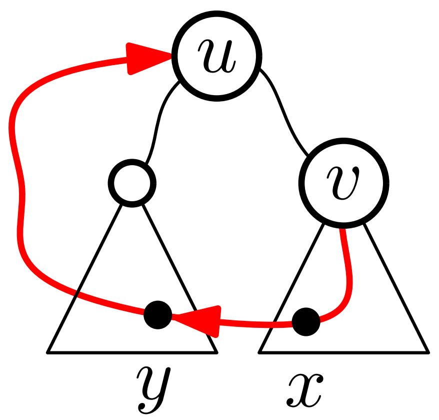


Proof of correctness

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 $y \notin T_v$.

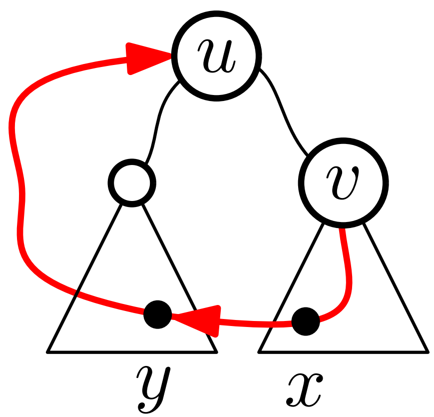


Proof of correctness

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.

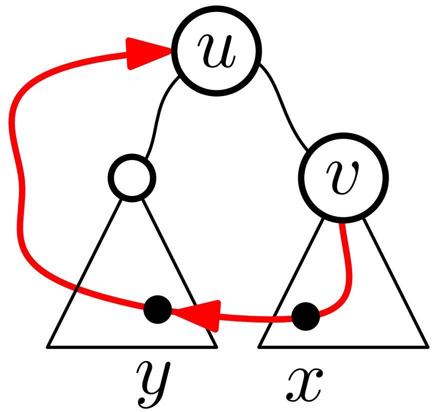


Proof of correctness

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

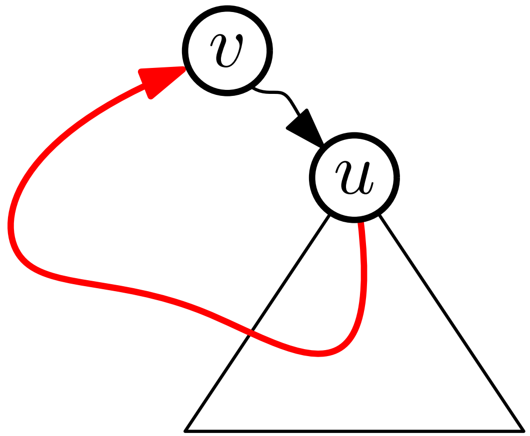
Claim: Let u be the first encountered head in postorder. $\eta(u) = exit(u)$.

- 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. $\eta(u) = exit(u)$.

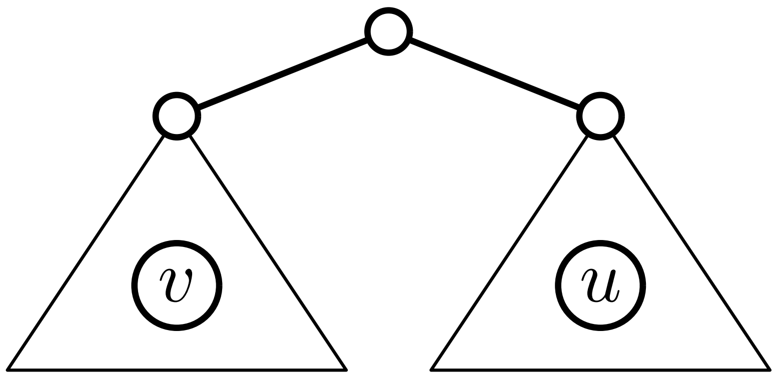
- Assume that there is a vertex v s.t. $\eta(v) = exit(u) < \eta(u)$.
- v cannot be an ancestor of u (otherwise $v \in C$ and u is not the head of C).



Proof of correctness

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

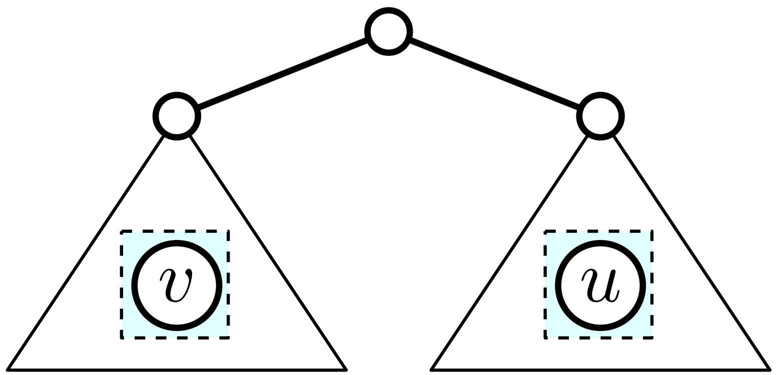
- Assume that there is a vertex v s.t. $\eta(v) = exit(u) < \eta(u)$.
- v cannot be an ancestor of u (otherwise $v \in C$ and u is not the head of C).



Proof of correctness

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

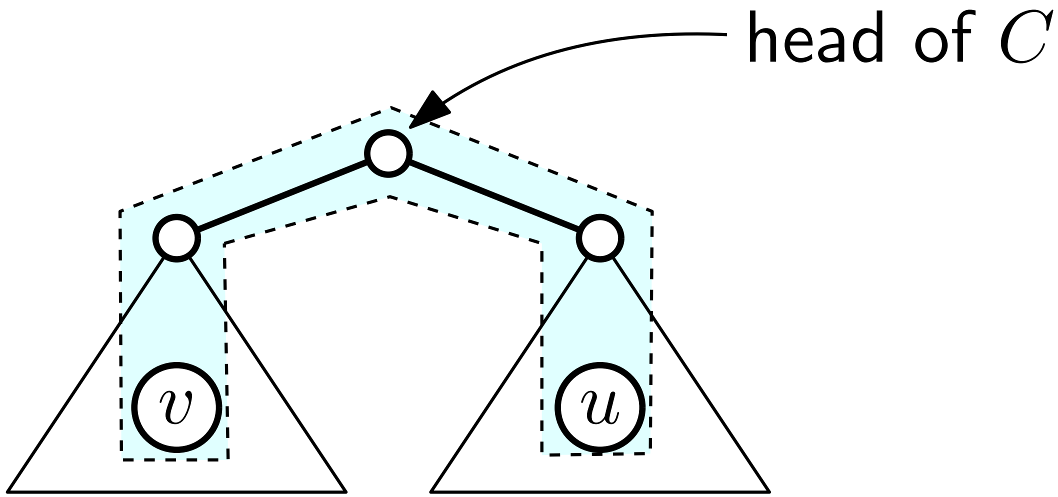
- Assume that there is a vertex v s.t. $\eta(v) = exit(u) < \eta(u)$.
- v cannot be an ancestor of u (otherwise $v \in C$ and u is not the head of C).
- If $v \in C$, then u and v are connected in $T[C] \implies$ the lowest common ancestor of u and v is in C.



Proof of correctness

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

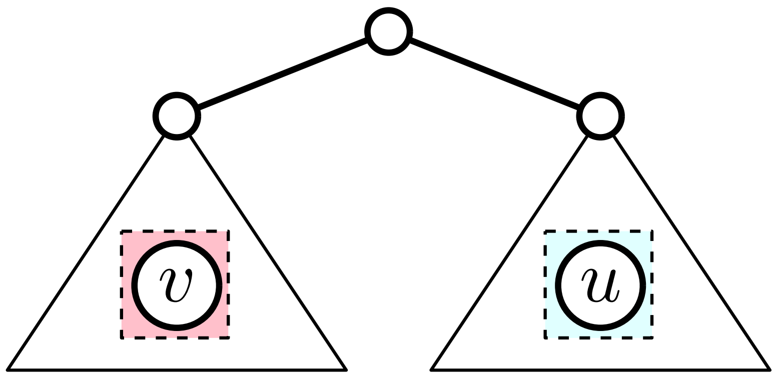
- Assume that there is a vertex v s.t. $\eta(v) = exit(u) < \eta(u)$.
- v cannot be an ancestor of u (otherwise $v \in C$ and u is not the head of C).
- If $v \in C$, then u and v are connected in $T[C] \implies$ the lowest common ancestor of u and v is in C. ↯



Proof of correctness

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

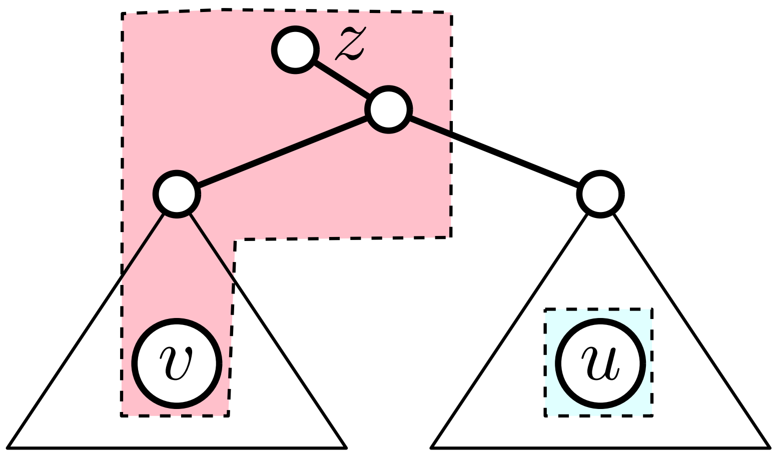
- Assume that there is a vertex v s.t. $\eta(v) = exit(u) < \eta(u)$.
- v cannot be an ancestor of u (otherwise $v \in C$ and u is not the head of C).
- If $v \in C$, then u and v are connected in $T[C] \implies$ the lowest common ancestor of u and v is in C. ↯
- If $v \in C' \neq C$ then the head z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

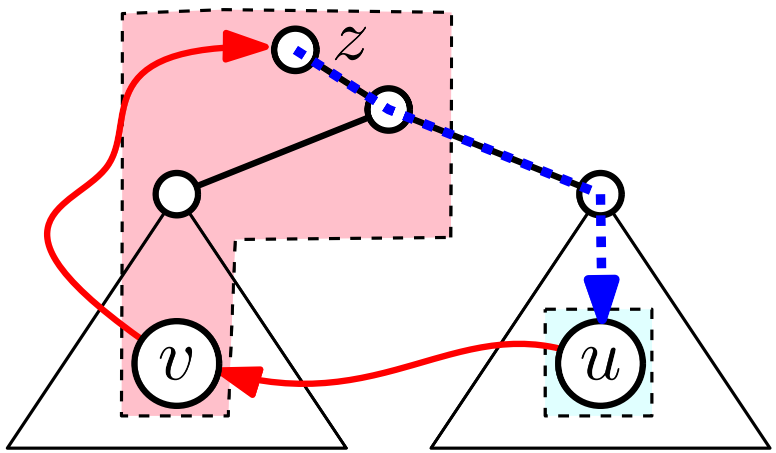
- Assume that there is a vertex v s.t. $\eta(v) = exit(u) < \eta(u)$.
- v cannot be an ancestor of u (otherwise $v \in C$ and u is not the head of C).
- If $v \in C$, then u and v are connected in $T[C] \implies$ the lowest common ancestor of u and v is in C. $\lightning$
- If $v \in C' \neq C$ then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

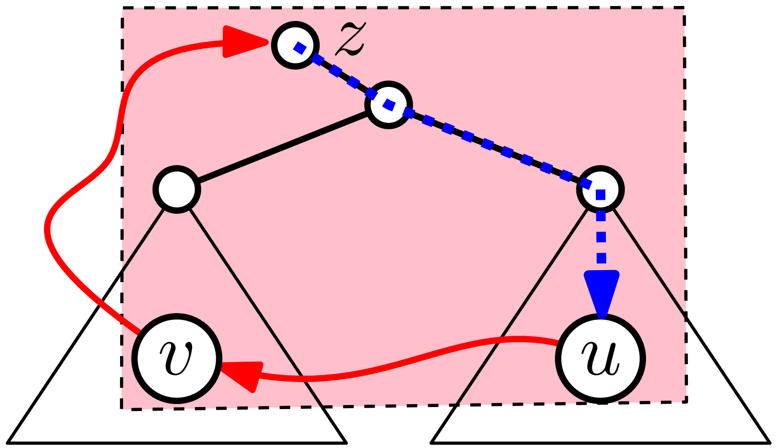
- Assume that there is a vertex v s.t. $\eta(v) = exit(u) < \eta(u)$.
- v cannot be an ancestor of u (otherwise $v \in C$ and u is not the head of C).
- If $v \in C$, then u and v are connected in $T[C] \implies$ the lowest common ancestor of u and v is in C. ↯
- If $v \in C' \neq C$ then the head z 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 $\leftarrow 0$; $T \leftarrow (\{u\}, \emptyset)$ $\quad$ $S \leftarrow$ Empty stack
- SCC(u)

SCC(u):

- $\eta(u) \leftarrow$ cnt; cnt $\leftarrow$ cnt $+1$; $exit(u) \leftarrow \eta(u)$ $\quad$ 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)