

COL 351 : ANALYSIS & DESIGN OF ALGORITHMS

LECTURE 12

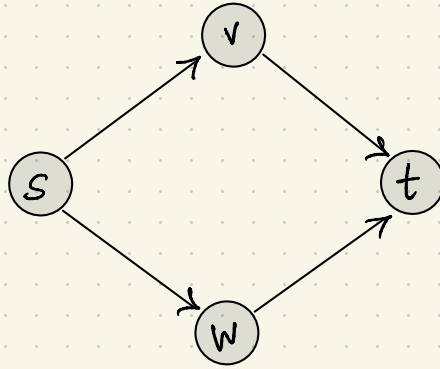
GRAPH ALGORITHMS I : STRONGLY CONNECTED COMPONENTS (CONTD.)

AUG 19, 2026

|

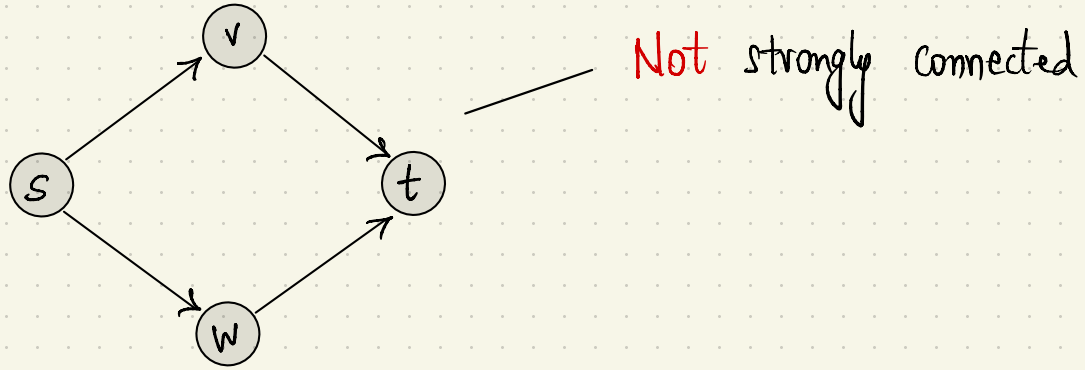
ROHIT VAISH

STRONGLY CONNECTED COMPONENTS



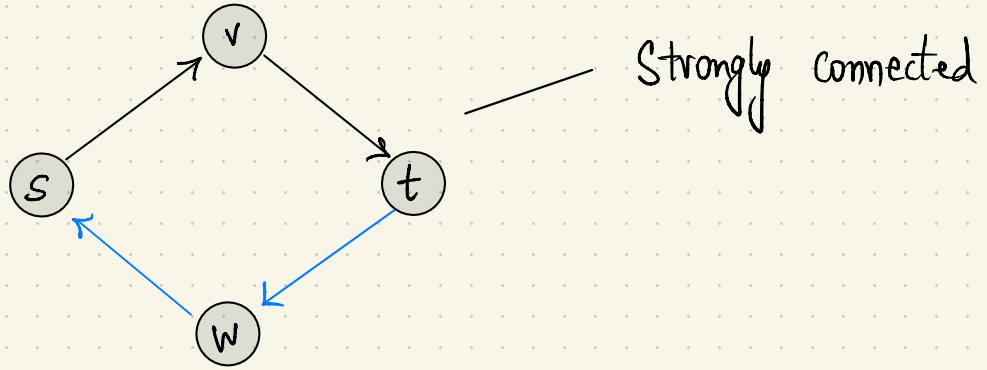
A directed graph is **strongly connected** if every vertex can be reached from every other vertex by a directed path.

STRONGLY CONNECTED COMPONENTS



A directed graph is **strongly connected** if every vertex can be reached from every other vertex by a directed path.

STRONGLY CONNECTED COMPONENTS



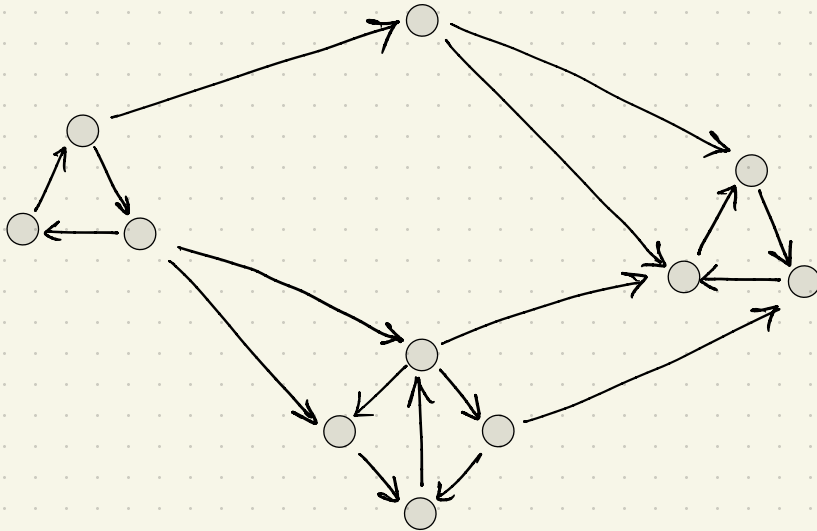
A directed graph is **strongly connected** if every vertex can be reached from every other vertex by a directed path.

STRONGLY CONNECTED COMPONENTS

A strongly connected component of a directed graph G is a **maximal** subgraph of G that is strongly connected.

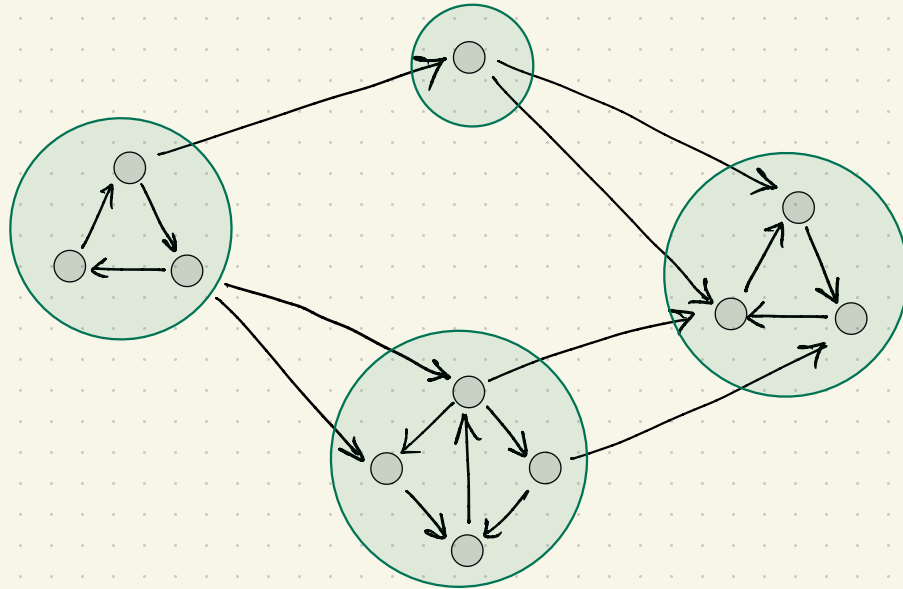
STRONGLY CONNECTED COMPONENTS

A strongly connected component of a directed graph G is a **maximal** subgraph of G that is strongly connected.



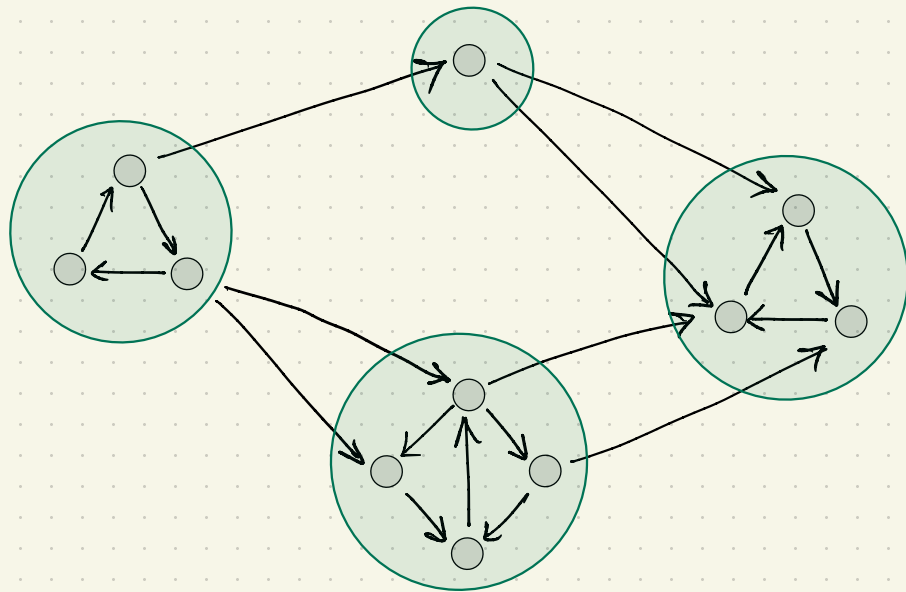
STRONGLY CONNECTED COMPONENTS

A strongly connected component of a directed graph G is a **maximal** subgraph of G that is strongly connected.



STRONGLY CONNECTED COMPONENTS

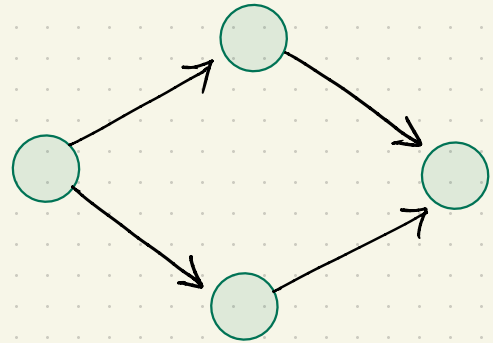
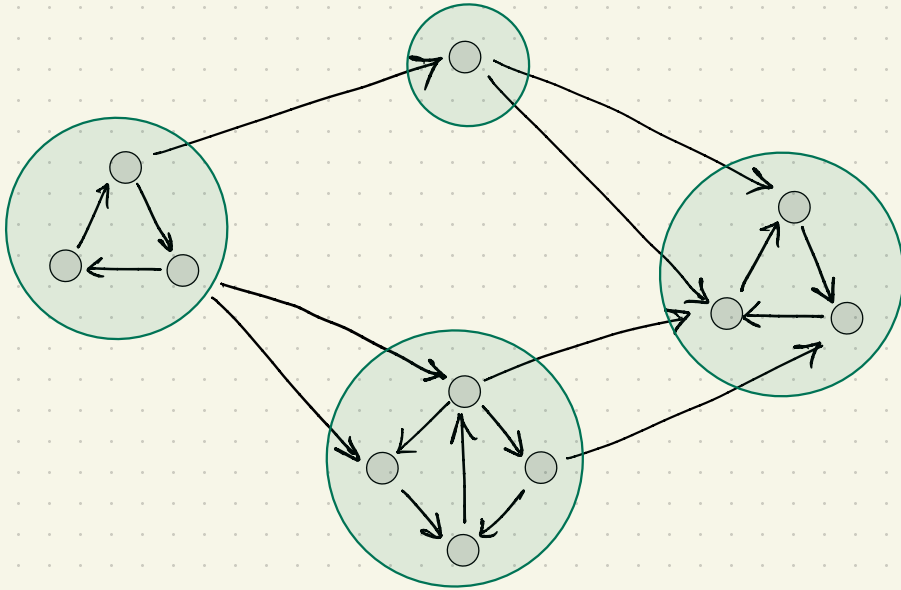
A strongly connected component of a directed graph G is a **maximal** subgraph of G that is strongly connected.



A DAG!

STRONGLY CONNECTED COMPONENTS

A strongly connected component of a directed graph G is a **maximal** subgraph of G that is strongly connected.

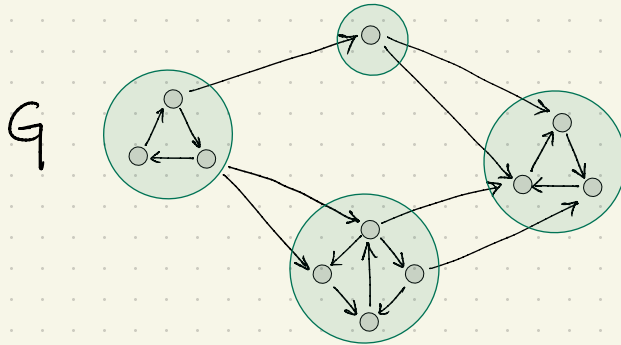


STRONGLY CONNECTED COMPONENTS

Theorem:

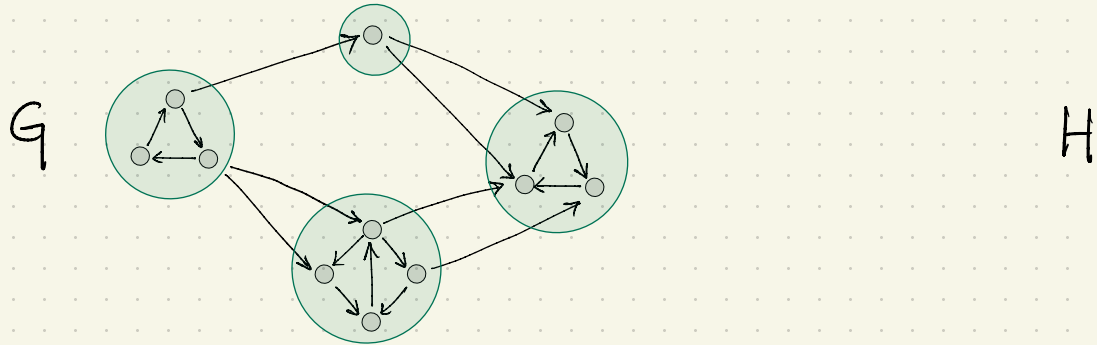
STRONGLY CONNECTED COMPONENTS

Theorem: Given a directed graph G



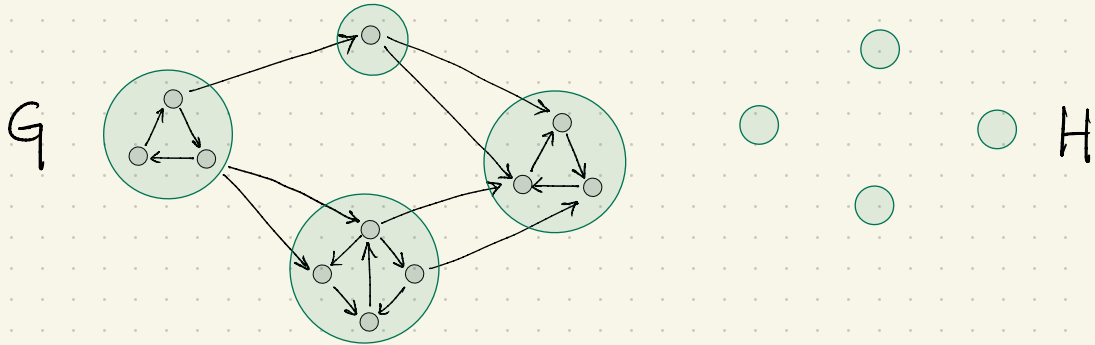
STRONGLY CONNECTED COMPONENTS

Theorem: Given a directed graph G , define a "meta" graph $H=(X,F)$



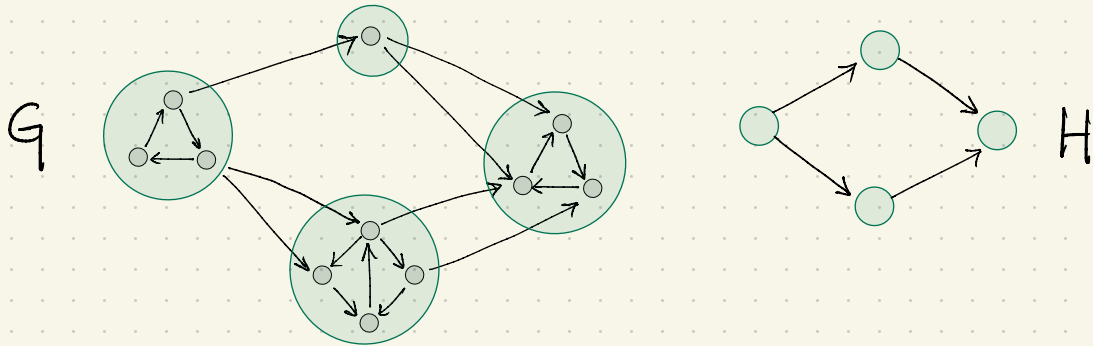
STRONGLY CONNECTED COMPONENTS

Theorem: Given a directed graph G , define a "meta" graph $H=(X,F)$ that has a vertex for every SCC of G



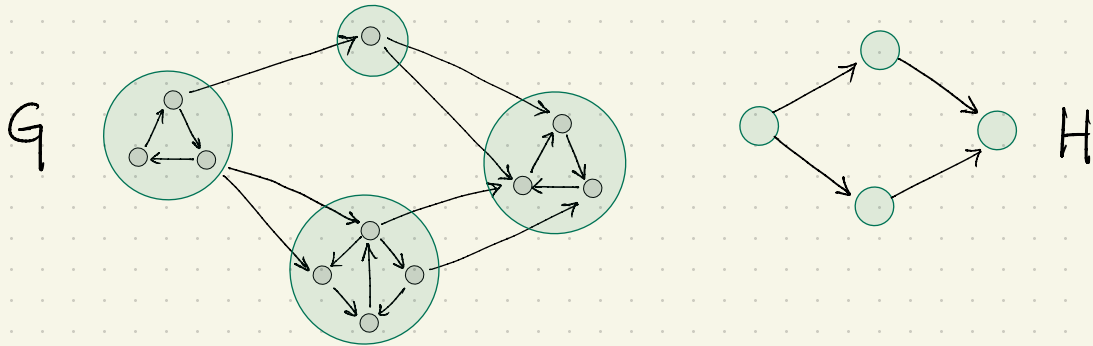
STRONGLY CONNECTED COMPONENTS

Theorem: Given a directed graph G , define a "meta" graph $H=(X,F)$ that has a vertex for every SCC of G , and an edge $(x,y) \in F$ if there is an edge from some vertex in SCC corresponding to x to some vertex in SCC corresponding to y .



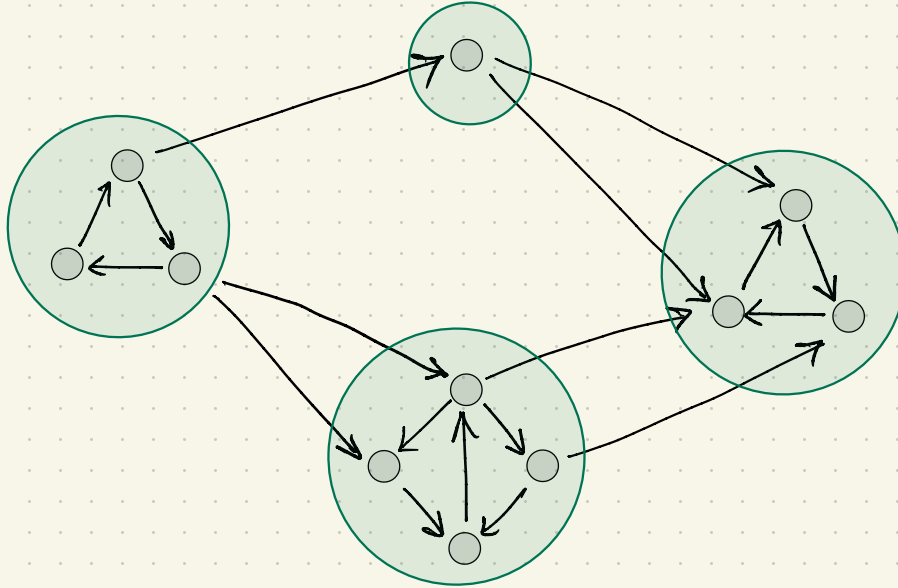
STRONGLY CONNECTED COMPONENTS

Theorem: Given a directed graph G , define a "meta" graph $H=(X,F)$ that has a vertex for every SCC of G , and an edge $(x,y) \in F$ if there is an edge from some vertex in SCC corresponding to x to some vertex in SCC corresponding to y . Then, H is a DAG.

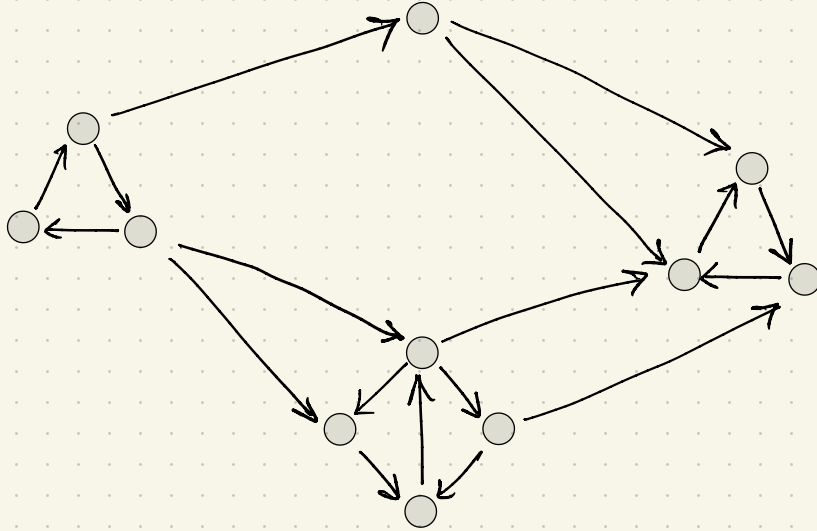


STRONGLY CONNECTED COMPONENTS via DFS

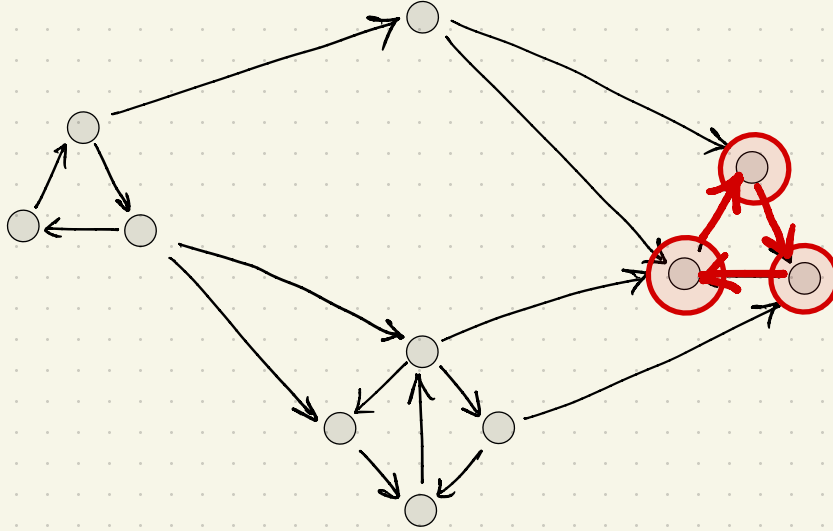
STRONGLY CONNECTED COMPONENTS via DFS



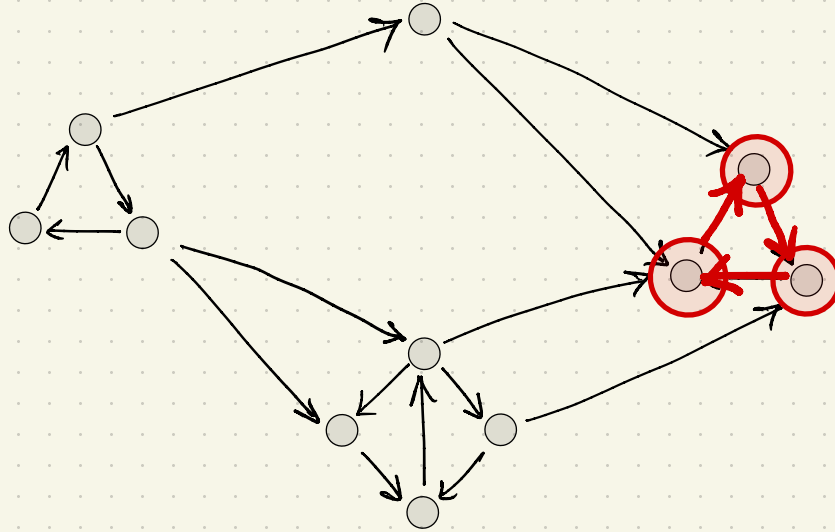
STRONGLY CONNECTED COMPONENTS via DFS



STRONGLY CONNECTED COMPONENTS via DFS

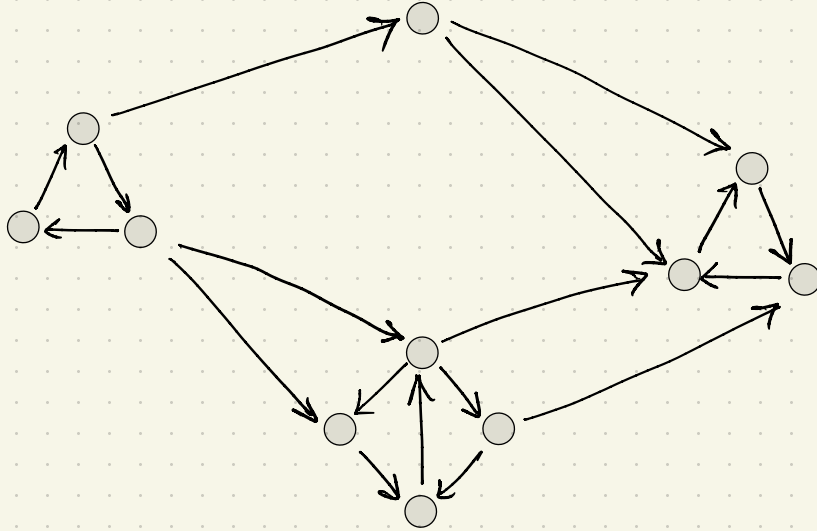


STRONGLY CONNECTED COMPONENTS via DFS



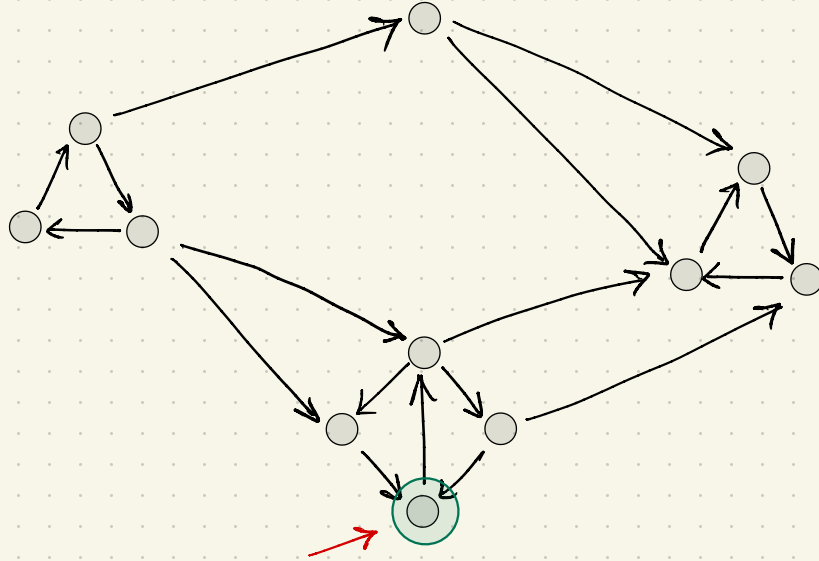
an SCC!

STRONGLY CONNECTED COMPONENTS via DFS



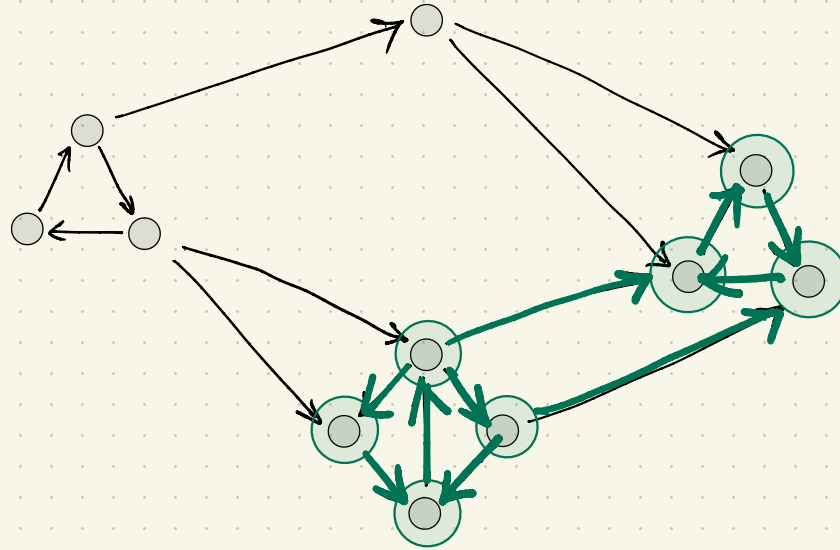
🤔 Maybe do a DFS from every node to identify all SCCs?

STRONGLY CONNECTED COMPONENTS via DFS



🤔 Maybe do a DFS from every node to identify all SCCs?

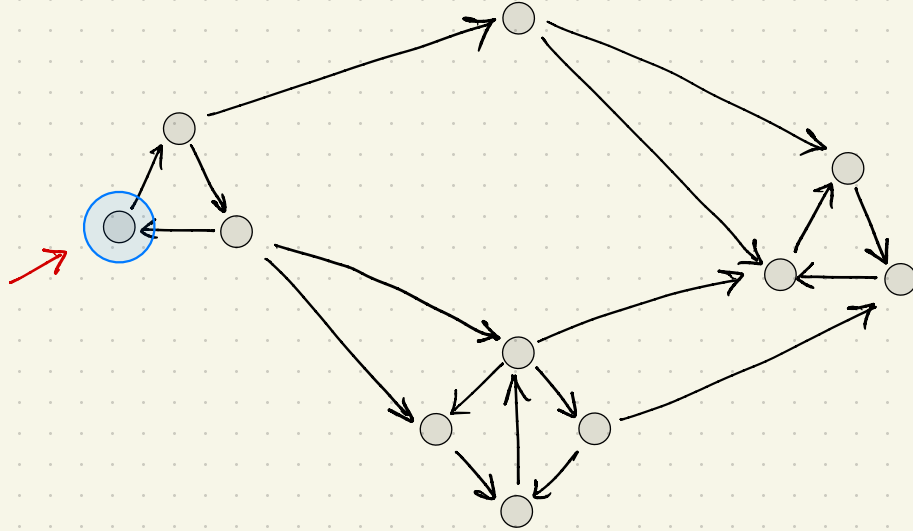
STRONGLY CONNECTED COMPONENTS via DFS



Union of SCCs

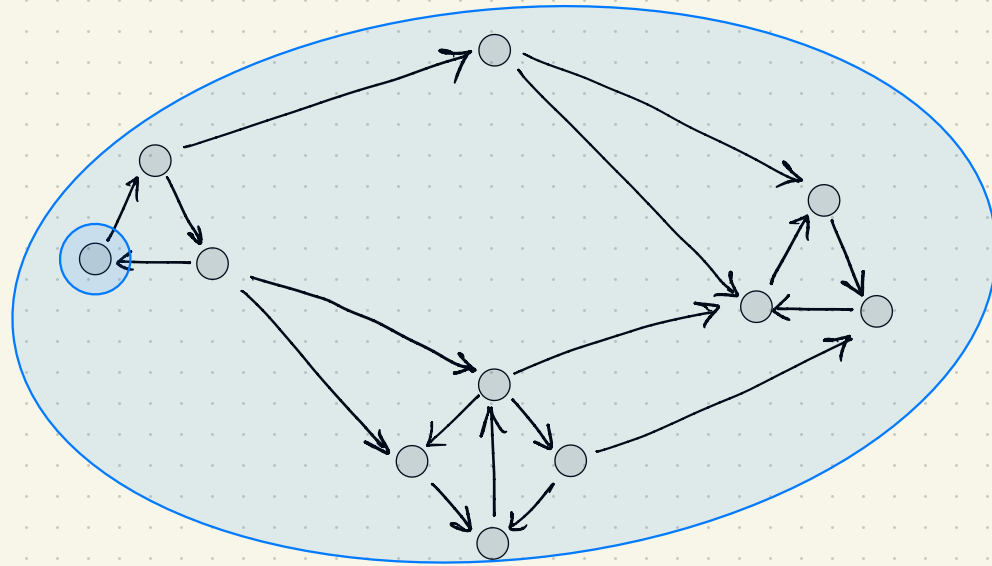
🤔 Maybe do a DFS from every node to identify all SCCs?

STRONGLY CONNECTED COMPONENTS via DFS



🤔 Maybe do a DFS from every node to identify all SCCs?

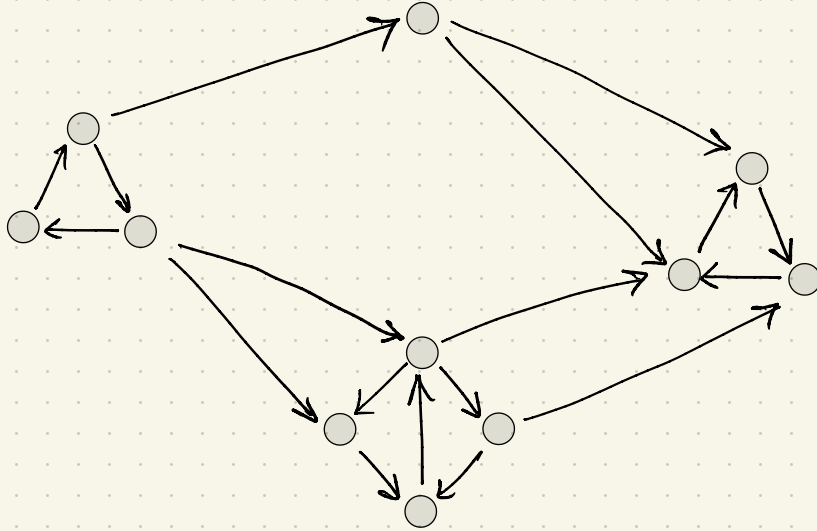
STRONGLY CONNECTED COMPONENTS via DFS



No information
at all!

🤔 Maybe do a DFS from every node to identify all SCCs?

STRONGLY CONNECTED COMPONENTS via DFS

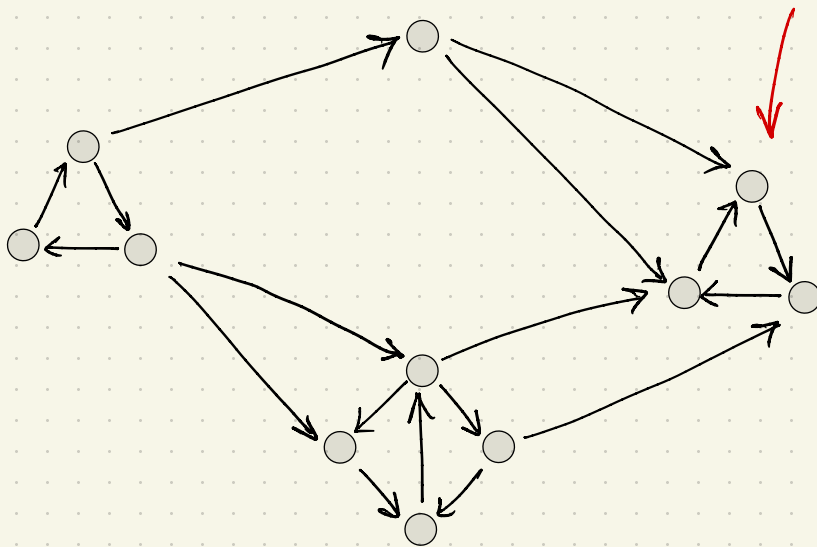


Starting point matters!

FIRST ATTEMPT

FIRST ATTEMPT

Recall: Starting here worked!



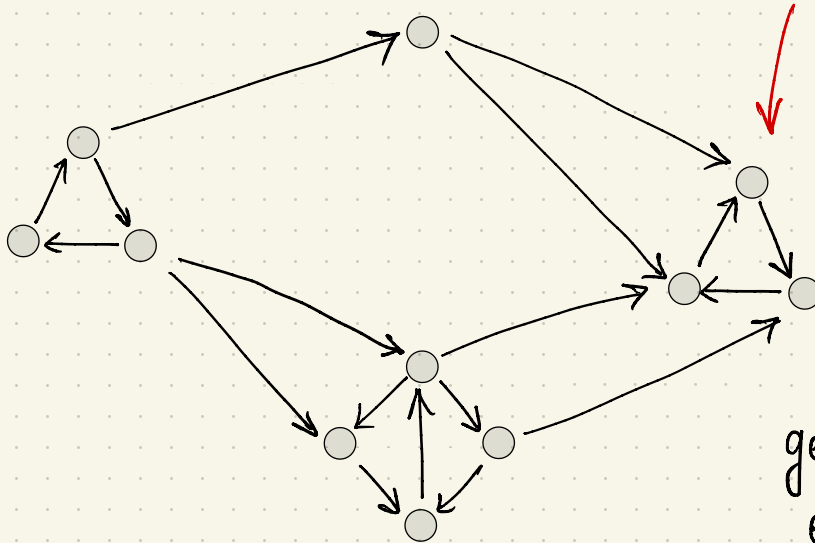
maybe it lies in the "sink" SCC?



Start DFS at the "last" vertex
(as per topological ordering algo)

FIRST ATTEMPT

Recall: Starting here worked!

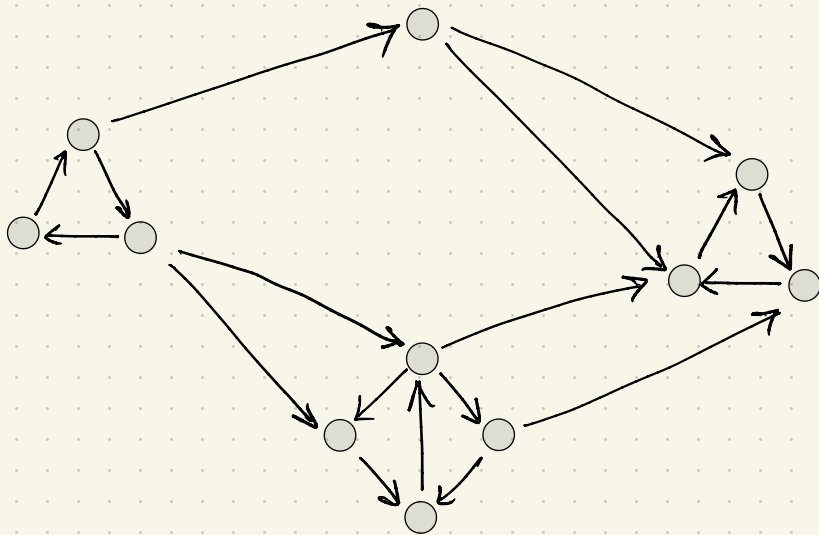


generates **some** ordering
even for cyclic graphs



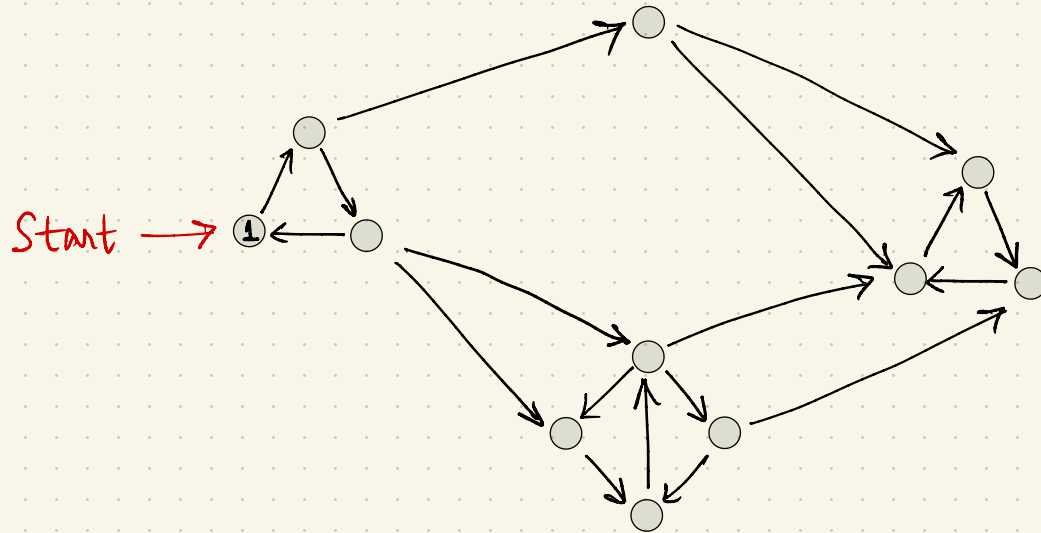
Start DFS at the "last" vertex
(as per **topological ordering algo**)

FIRST ATTEMPT

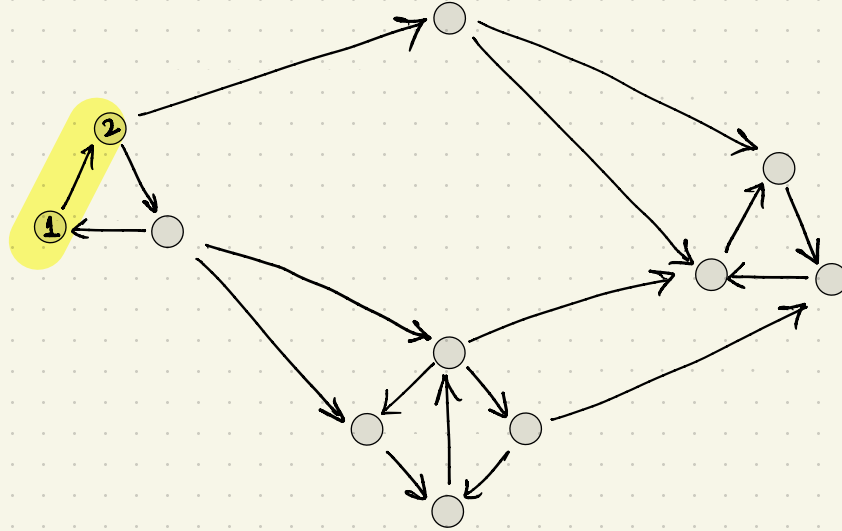


Let's run topological ordering (via DFS) and see what happens.

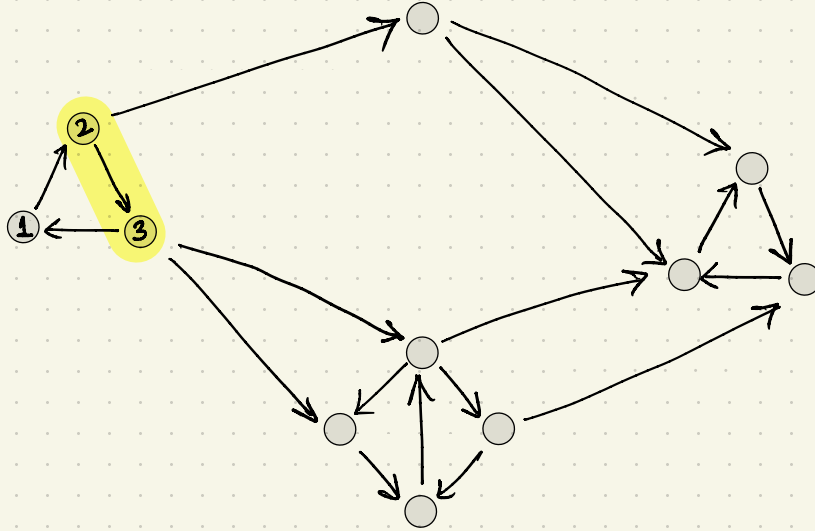
FIRST ATTEMPT



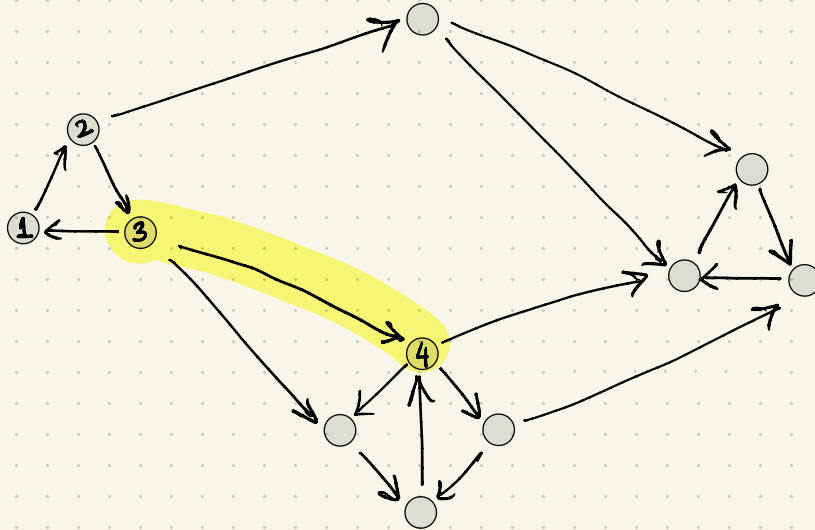
FIRST ATTEMPT



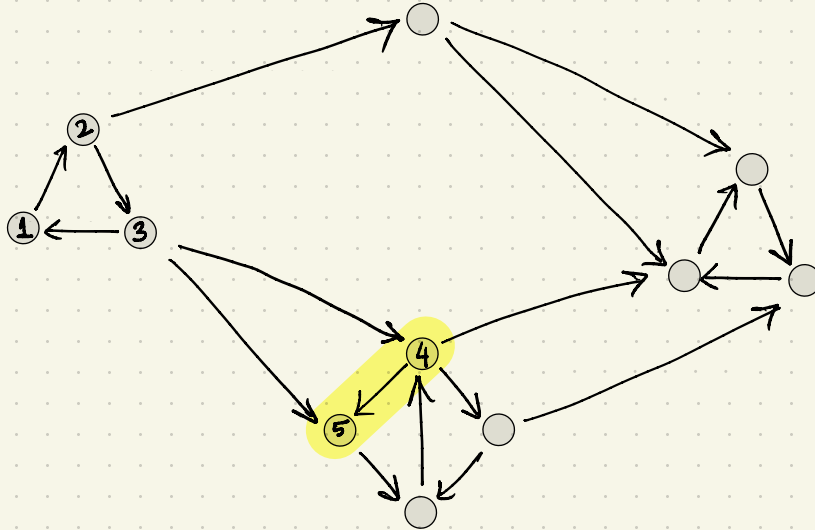
FIRST ATTEMPT



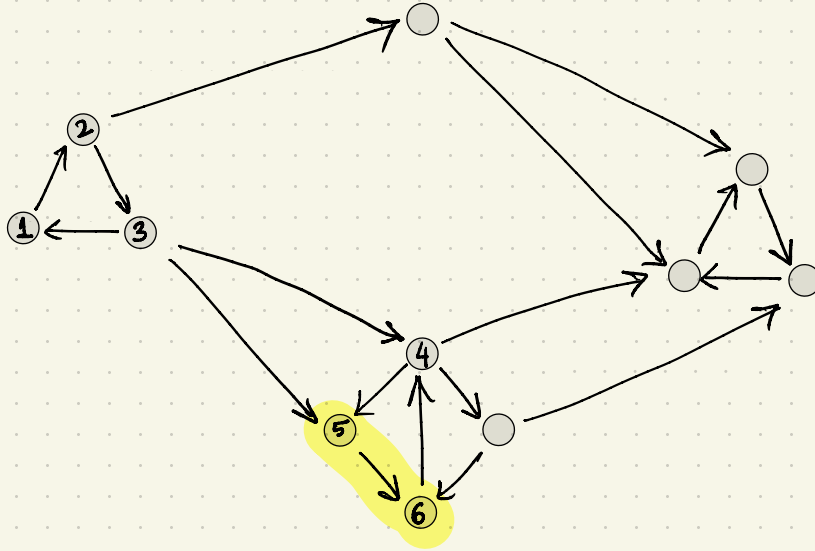
FIRST ATTEMPT



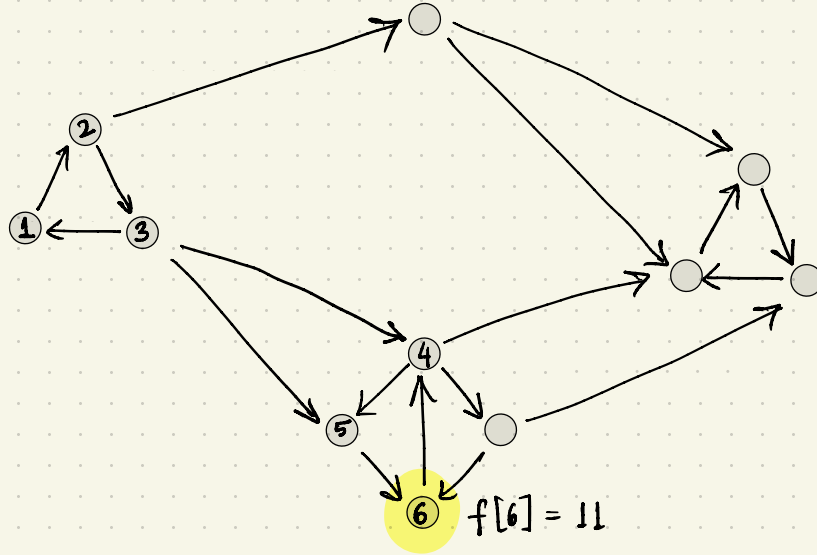
FIRST ATTEMPT



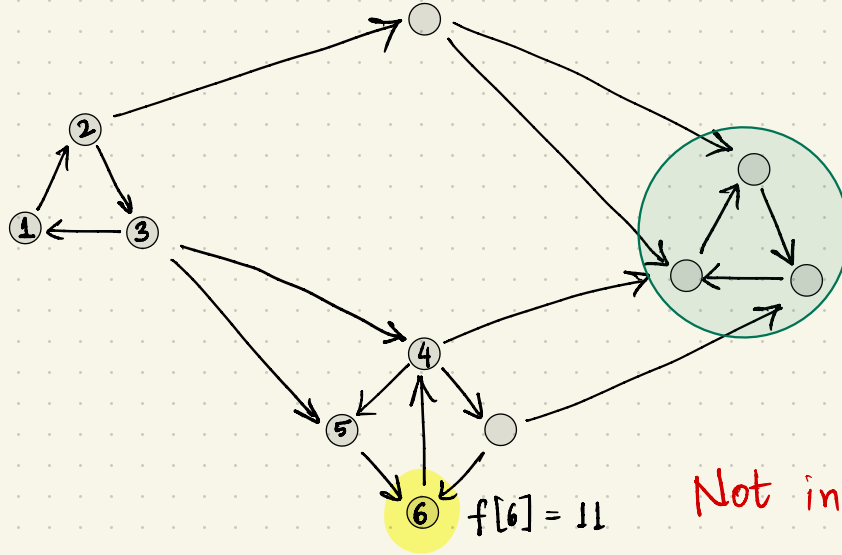
FIRST ATTEMPT



FIRST ATTEMPT

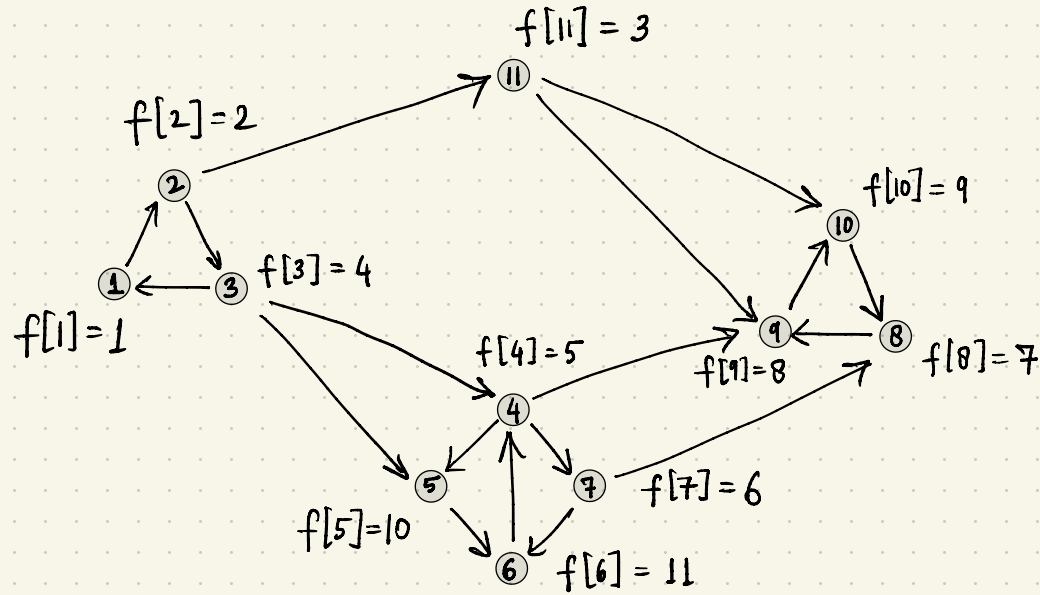


FIRST ATTEMPT

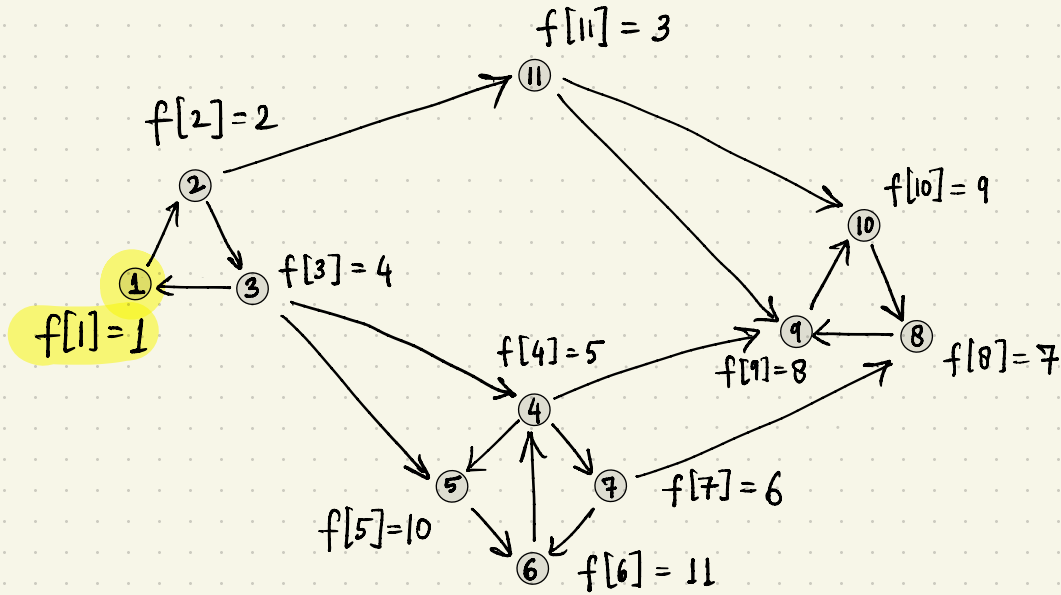


Not in "sink" SCC ☹️

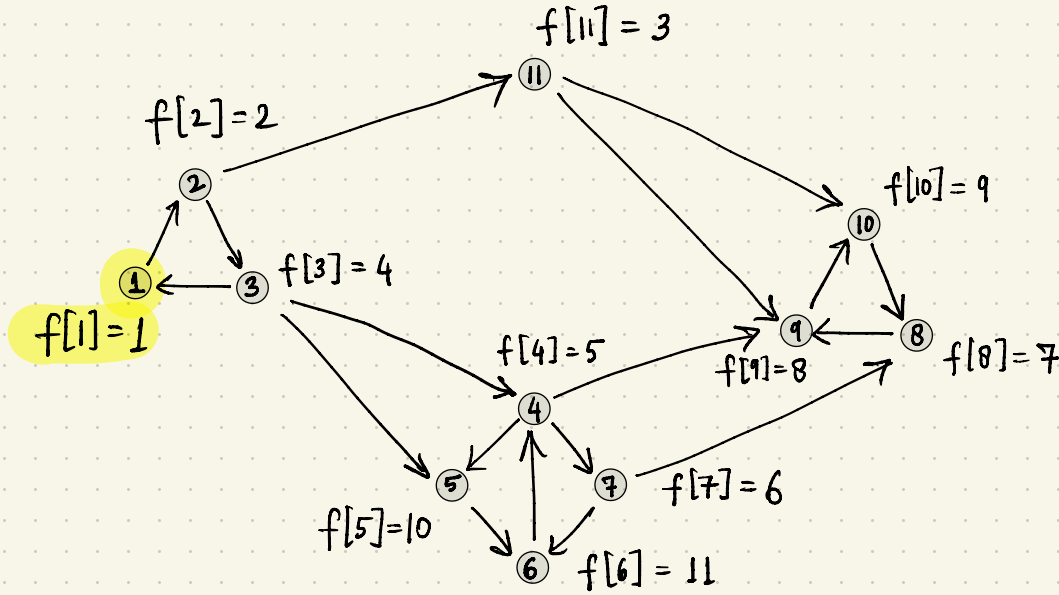
FIRST ATTEMPT



FIRST ATTEMPT

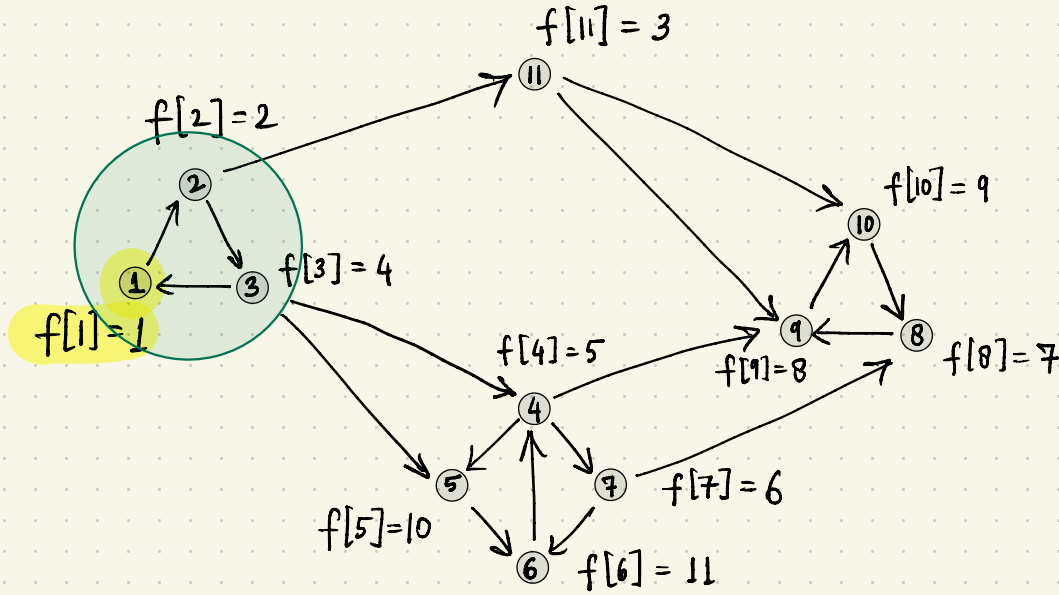


FIRST ATTEMPT



Perhaps the "smallest-label" vertex is always in "source" SCC?
(as per topological ordering)

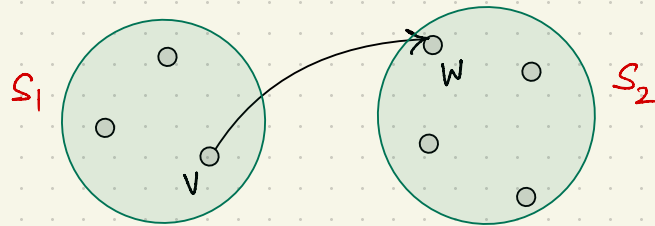
FIRST ATTEMPT



Perhaps the "smallest-label" vertex is always in "source" SCC?
(as per topological ordering)

KEY OBSERVATION

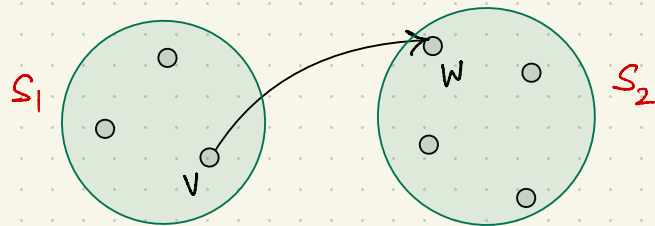
Theorem: Let f be the labeling of directed graph G generated by the topological ordering algorithm on G (arbitrary ordering of vertices). Let S_1, S_2 be two "adjacent" SCCs of G , i.e., there is an edge (v, w) with $v \in S_1$ and $w \in S_2$.



KEY OBSERVATION

Theorem: Let f be the labeling of directed graph G generated by the topological ordering algorithm on G (arbitrary ordering of vertices). Let S_1, S_2 be two "adjacent" SCCs of G , i.e., there is an edge (v, w) with $v \in S_1$ and $w \in S_2$. Then,

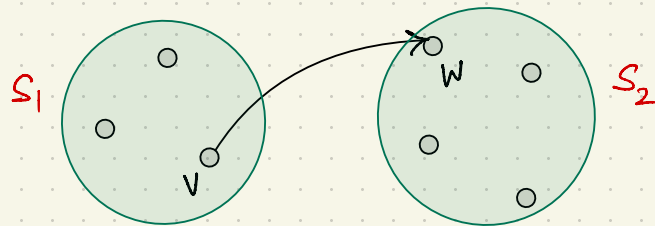
$$\min_{x \in S_1} f(x) < \min_{y \in S_2} f(y)$$



KEY OBSERVATION

Theorem: Let f be the labeling of directed graph G generated by the topological ordering algorithm on G (arbitrary ordering of vertices). Let S_1, S_2 be two "adjacent" SCCs of G , i.e., there is an edge (v, w) with $v \in S_1$ and $w \in S_2$. Then,

$$\min_{x \in S_1} f(x) < \min_{y \in S_2} f(y)$$



recursive call of some vertex in S_1 finishes after that of all vertices in S_2 .

KEY OBSERVATION

Theorem: Let f be the labeling of directed graph G generated by the topological ordering algorithm on G (arbitrary ordering of vertices). Let S_1, S_2 be two "adjacent" SCCs of G , i.e., there is an edge (v, w) with $v \in S_1$ and $w \in S_2$. Then,

$$\min_{x \in S_1} f(x) < \min_{y \in S_2} f(y).$$

Generalization of "every DAG has a topological ordering".

KEY OBSERVATION

Theorem: Let f be the labeling of directed graph G generated by the topological ordering algorithm on G (arbitrary ordering of vertices). Let S_1, S_2 be two "adjacent" SCCs of G , i.e., there is an edge (v, w) with $v \in S_1$ and $w \in S_2$. Then,

$$\min_{x \in S_1} f(x) < \min_{y \in S_2} f(y).$$

Corollary: The vertex v with $f[v] = 1$ must lie in source SCC.

Proof:

Proof: Case I: Topological algo. discovers some vertex in S_1 before S_2 .

Case II: Topological algo. discovers some vertex in S_2 before S_1 .

Proof: Case I: Topological algo. discovers some vertex in S_1 before S_2 .

All vertices in S_2 reachable from some vertex in S_1 .

Case II: Topological algo. discovers some vertex in S_2 before S_1 .

Proof: Case I: Topological algo. discovers some vertex in S_1 before S_2 .

All vertices in S_2 reachable from some vertex in S_1 .

$\Rightarrow$ Recursive calls for all vertices in S_2 finish before that of some vertex in S_1 .

Case II: Topological algo. discovers some vertex in S_2 before S_1 .

Proof: Case I: Topological algo. discovers some vertex in S_1 before S_2 .

All vertices in S_2 reachable from some vertex in S_1 .

$\Rightarrow$ Recursive calls for all vertices in S_2 finish before that of some vertex in S_1 .

Case II: Topological algo. discovers some vertex in S_2 before S_1 .

Recall: SCC meta-graph is acyclic.

Proof: Case I: Topological algo. discovers some vertex in S_1 before S_2 .

All vertices in S_2 reachable from some vertex in S_1 .

$\Rightarrow$ Recursive calls for all vertices in S_2 finish before that of some vertex in S_1 .

Case II: Topological algo. discovers some vertex in S_2 before S_1 .

Recall: SCC meta-graph is acyclic.

S_1 is unreachable from S_2 .

Proof: Case I: Topological algo. discovers some vertex in S_1 before S_2 .

All vertices in S_2 reachable from some vertex in S_1 .

$\Rightarrow$ Recursive calls for all vertices in S_2 finish before that of some vertex in S_1 .

Case II: Topological algo. discovers some vertex in S_2 before S_1 .

Recall: SCC meta-graph is acyclic.

S_1 is unreachable from S_2 .

$\Rightarrow$ Recursive calls for all vertices in S_2 finish before that of every vertex in S_1 .

Proof: Case I: Topological algo. discovers some vertex in S_1 before S_2 .

All vertices in S_2 reachable from some vertex in S_1 .

$\Rightarrow$ Recursive calls for all vertices in S_2 finish before that of some vertex in S_1 .

Case II: Topological algo. discovers some vertex in S_2 before S_1 .

Recall: SCC meta-graph is acyclic.

S_1 is unreachable from S_2 .

$\Rightarrow$ Recursive calls for all vertices in S_2 finish before that of every vertex in S_1 . ▣

What we have : A way of identifying a vertex in **source** SCC.

What we want : A way of identifying a vertex in **sink** SCC.

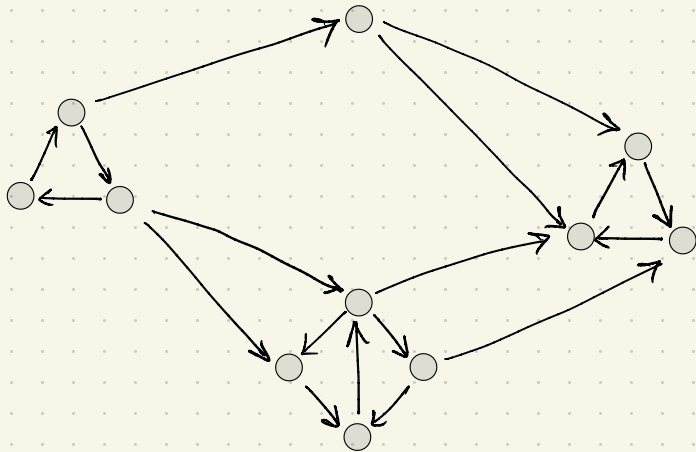
What's the fix :

What we have : A way of identifying a vertex in **source** SCC.

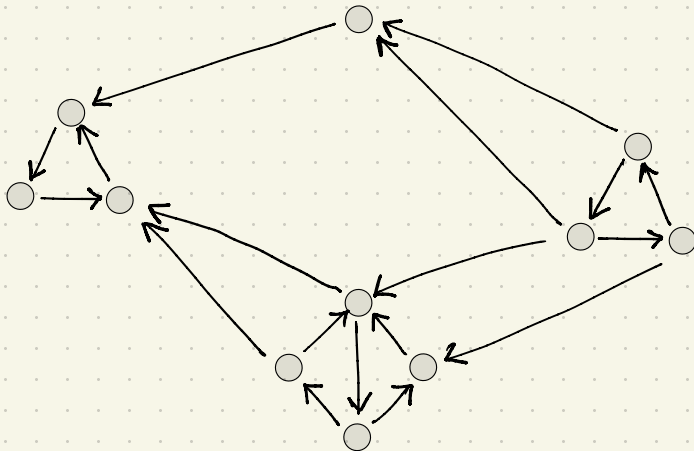
What we want : A way of identifying a vertex in **sink** SCC.

What's the fix : **Reverse** the graph !

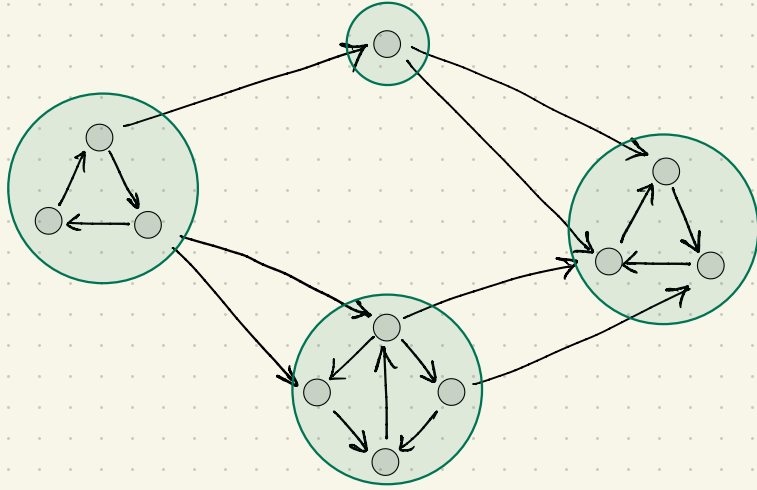
G



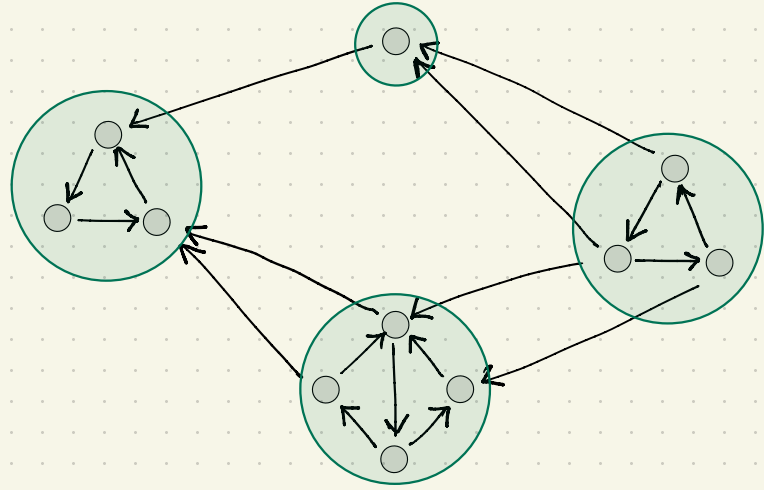
GeV



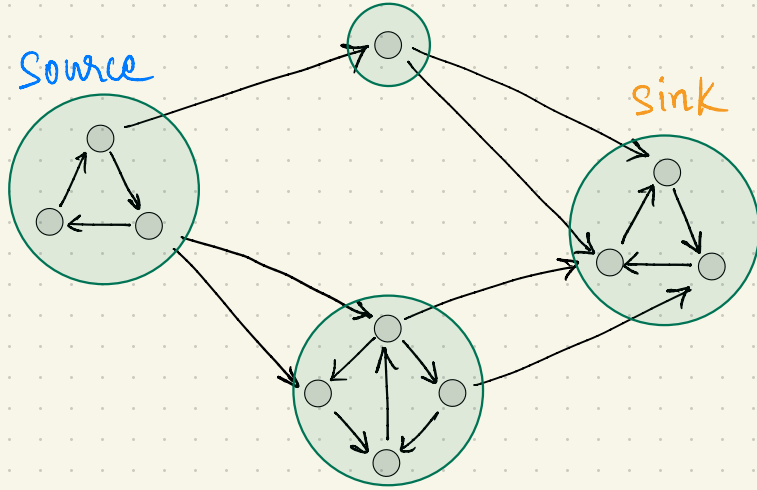
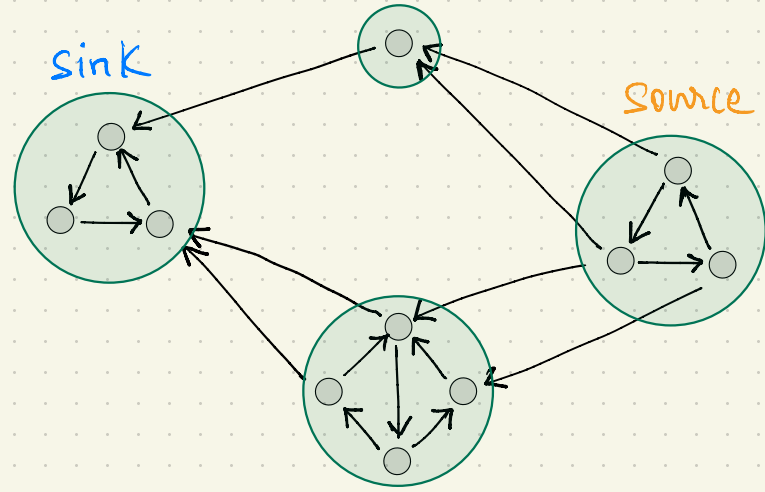
G



G^{rev}

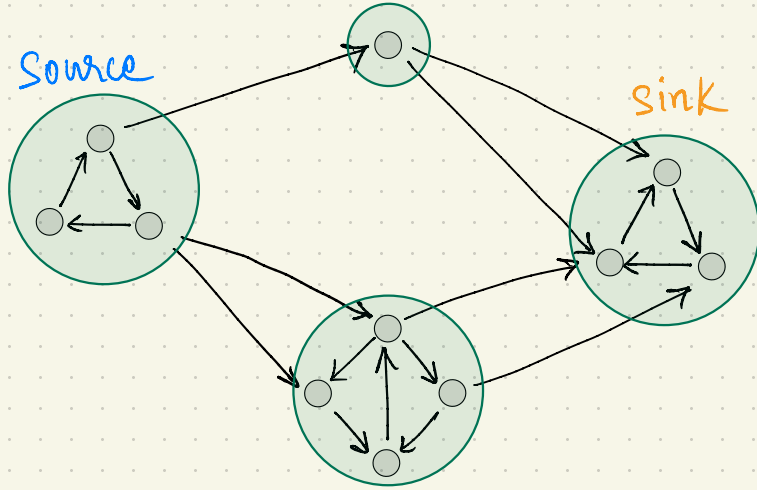


SCCs stay the same!

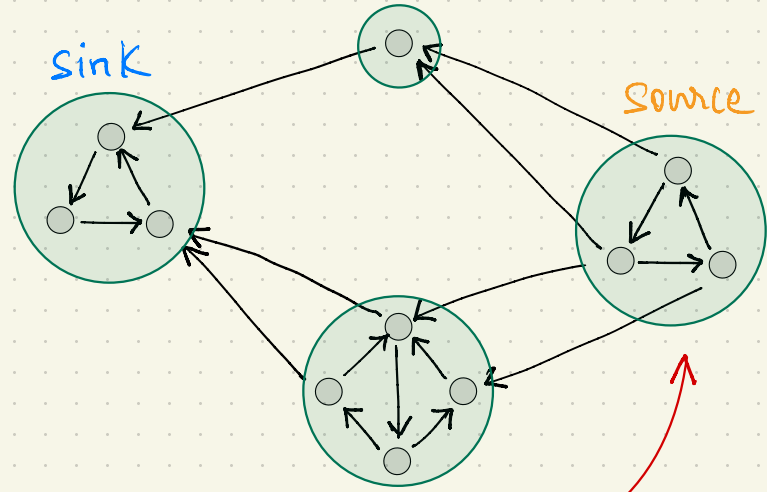
G  G^{rev} 

Source / sink in meta graph of G
Sink / Source " " " G^{rev}

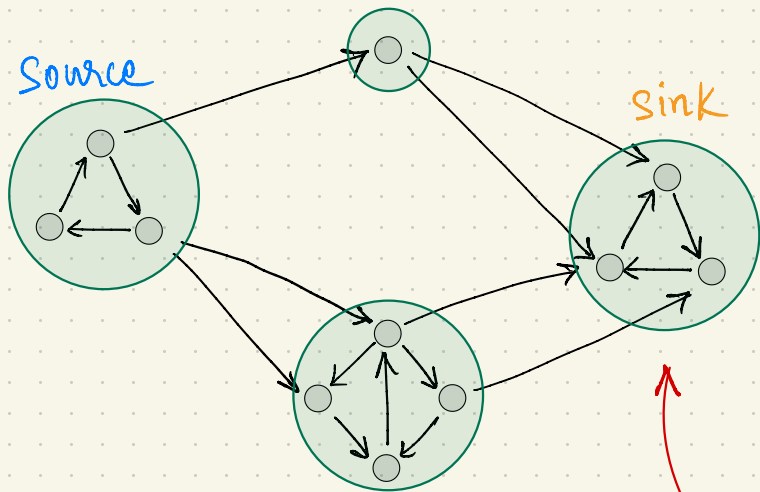
G



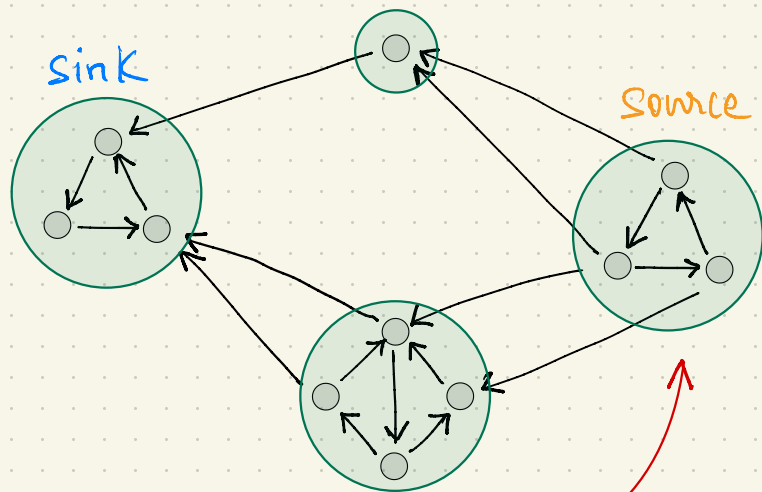
G^{rev}



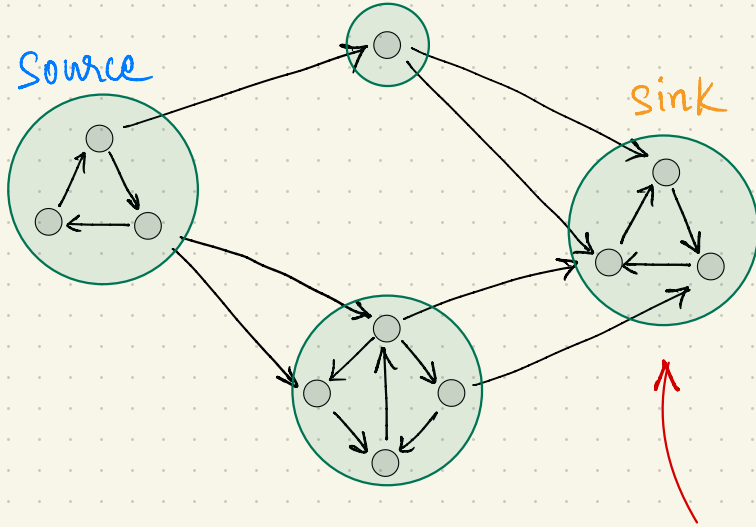
identify a vertex here

G 

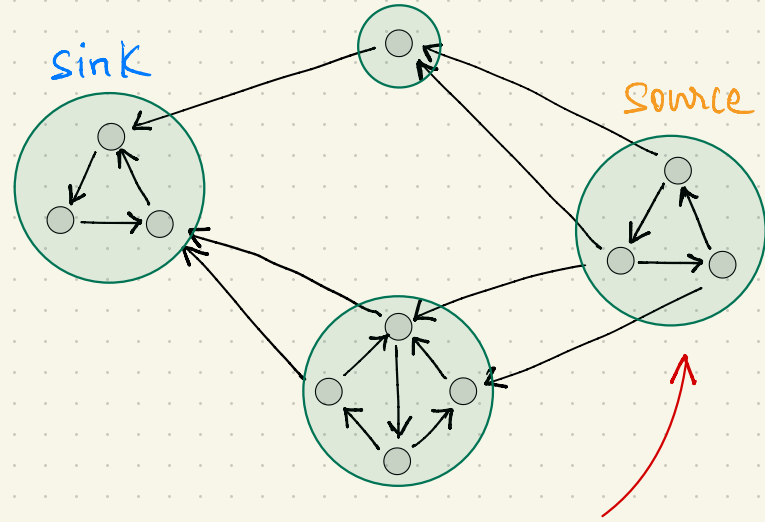
equivalently, here

 G^{rev} 

identify a vertex here

G 

equivalently, here

 G^{rev} 

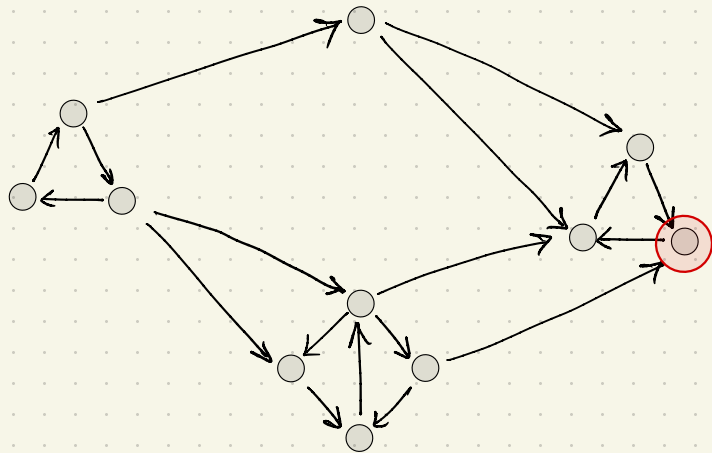
identify a vertex here



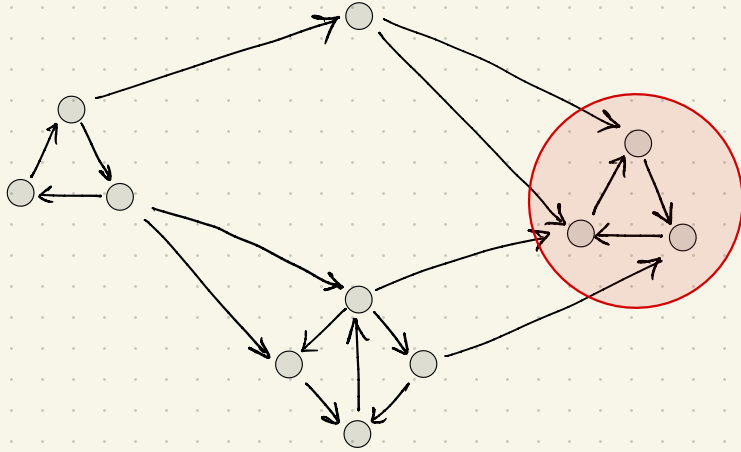
"Pluck out" a sink SCC of G and work our way backward.

(similar to topological ordering algo.)

G

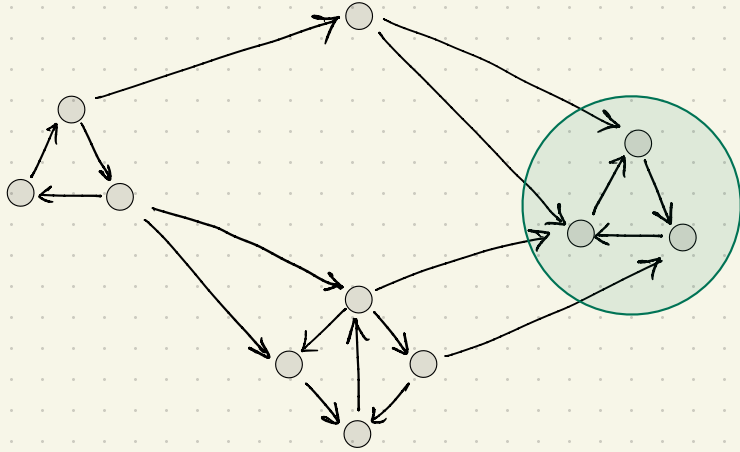


G



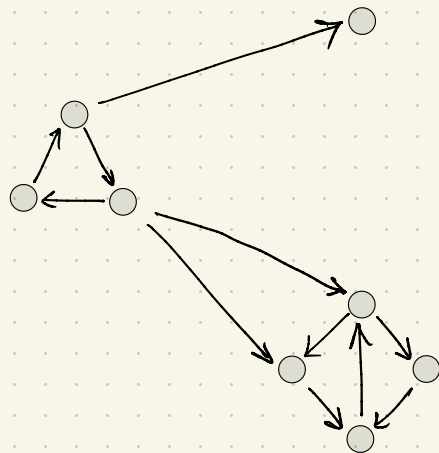
discover sink SCC via DFS

G

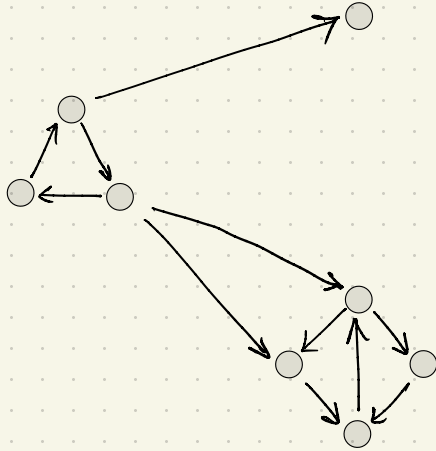


mark sink SCC as explored

G

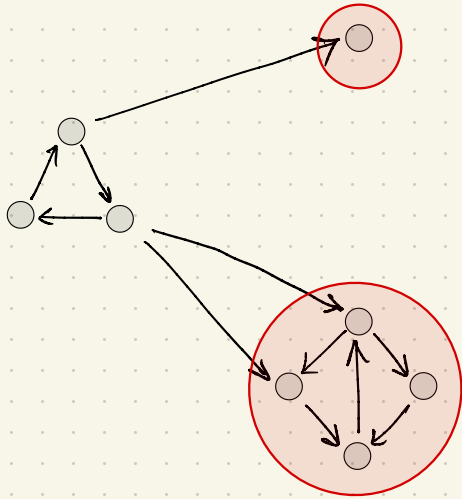


G



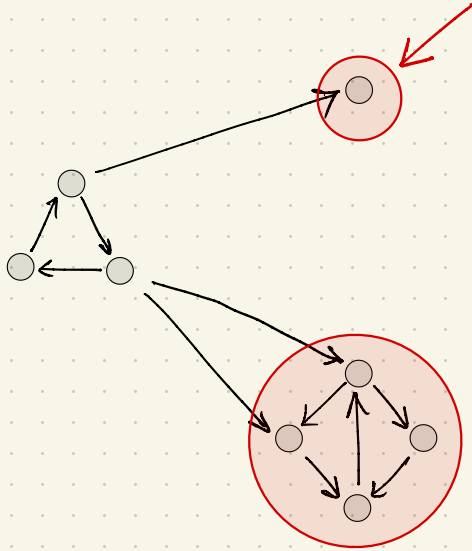
key observation $\Rightarrow$
among **unexplored** vertices, the "min f" label
should lie in one of the new sink SCCs

G



Key observation $\Rightarrow$
among **unexplored** vertices, the "min f " label
should lie in one of the new sink SCCs

G

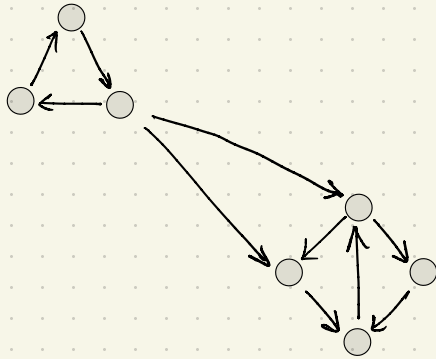


Key observation $\Rightarrow$

among **unexplored** vertices, the "min f " label should lie in one of the new sink SCCs

"Pluck out" the sink SCC and repeat!

G

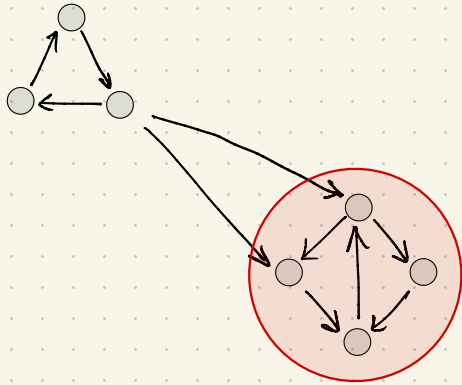


key observation $\Rightarrow$

among **unexplored** vertices, the "min f " label should lie in one of the new sink SCCs

"Pluck out" the sink SCC and repeat!

G

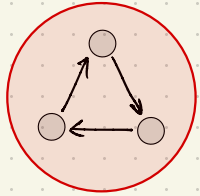


Key observation $\Rightarrow$

among **unexplored** vertices, the "min f " label should lie in one of the new sink SCCs

"Pluck out" the sink SCC and repeat!

G



Key observation $\Rightarrow$

among **unexplored** vertices, the "min f" label should lie in one of the new sink SCCs

"Pluck out" the sink SCC and repeat!

KOSARAJU'S ALGORITHM

KOSARAJU'S ALGORITHM

A *two-pass* algorithm to compute SCCs in $O(m+n)$ time.

KOSARAJU'S ALGORITHM

A **two-pass** algorithm to compute SCCs in $O(m+n)$ time.

- * Once on G^{rev} to find the "magical" labeling f
- * Once on G to find the SCCs using f .

KOSARAJU'S ALGORITHM

input : a directed graph $G = (V, E)$

output : all SCCs of G

KOSARAJU'S ALGORITHM

input: a directed graph $G = (V, E)$

output: all SCCs of G

① $G^{\text{rev}} :=$ reverse edges of G
mark all vertices of G^{rev} as **unexplored**

KOSARAJU's ALGORITHM

input: a directed graph $G = (V, E)$

output: all SCCs of G

① $G^{\text{rev}} :=$ reverse edges of G
mark all vertices of G^{rev} as **unexplored**

② Compute labeling f via topological
ordering algo on G^{rev}

KOSARAJU'S ALGORITHM

input: a directed graph $G = (V, E)$

output: all SCCs of G

- ① $G^{\text{rev}} :=$ reverse edges of G
mark all vertices of G^{rev} as **unexplored**
- ② Compute labeling f via topological
ordering algo on G^{rev}
- ③ mark all vertices of G as **unexplored**
global **numSCC** $:= 0$

KOSARAJU'S ALGORITHM

input: a directed graph $G = (V, E)$

output: all SCCs of G

- ① $G^{\text{rev}} :=$ reverse edges of G
mark all vertices of G^{rev} as **unexplored**
- ② Compute labeling f via topological
ordering algo on G^{rev}
- ③ mark all vertices of G as **unexplored**
global **numSCC** $:= 0$
for each v in **increasing order** of $f[v]$
|

KOSARAJU'S ALGORITHM

input: a directed graph $G = (V, E)$

output: all SCCs of G

① $G^{\text{rev}} :=$ reverse edges of G
mark all vertices of G^{rev} as **unexplored**

② Compute labeling f via topological
ordering algo on G^{rev}

③ mark all vertices of G as **unexplored**
global **numSCC** $:= 0$

for each v in increasing order of $f[v]$

 if v is **unexplored**
 |
 |

KOSARAJU'S ALGORITHM

input: a directed graph $G = (V, E)$

output: all SCCs of G

① $G^{\text{rev}} :=$ reverse edges of G
mark all vertices of G^{rev} as **unexplored**

② Compute labeling f via topological
ordering algo on G^{rev}

③ mark all vertices of G as **unexplored**
global **numSCC** $:= 0$

for each v in increasing order of $f[v]$

 if v is **unexplored**
 increment **numSCC** by 1

KOSARAJU'S ALGORITHM

input: a directed graph $G = (V, E)$

output: all SCCs of G

① $G^{\text{rev}} :=$ reverse edges of G
mark all vertices of G^{rev} as **unexplored**

② Compute labeling f via topological ordering algo on G^{rev}

③ mark all vertices of G as **unexplored**
global **numSCC** $:= 0$

for each v in increasing order of $f[v]$

 if v is **unexplored**
 increment **numSCC** by 1
 DFS-SCC(G, v)

KOSARAJU'S ALGORITHM

input: a directed graph $G = (V, E)$

output: all SCCs of G

DFS - SCC (G, v)

① $G^{\text{rev}} :=$ reverse edges of G
mark all vertices of G^{rev} as **unexplored**

② Compute labeling f via topological
ordering algo on G^{rev}

③ mark all vertices of G as **unexplored**
global **numSCC** := 0

for each v in increasing order of $f[v]$

 if v is **unexplored**
 increment **numSCC** by 1
 DFS - SCC (G, v)

KOSARAJU'S ALGORITHM

input: a directed graph $G = (V, E)$

output: all SCCs of G

① $G^{\text{rev}} :=$ reverse edges of G
mark all vertices of G^{rev} as **unexplored**

② Compute labeling f via topological
ordering algo on G^{rev}

③ mark all vertices of G as **unexplored**
global **numSCC** := 0

for each v in increasing order of $f[v]$

 if v is **unexplored**
 increment **numSCC** by 1
 DFS-SCC(G, v)

DFS-SCC(G, v)

// performs DFS and assigns
SCC values

KOSARAJU'S ALGORITHM

input: a directed graph $G = (V, E)$

output: all SCCs of G

① $G^{\text{rev}} :=$ reverse edges of G
mark all vertices of G^{rev} as **unexplored**

② Compute labeling f via topological ordering algo on G^{rev}

③ mark all vertices of G as **unexplored**
global **numSCC** := 0

for each v in increasing order of $f[v]$

 if v is **unexplored**
 increment **numSCC** by 1
 DFS-SCC(G, v)

DFS-SCC(G, v)

// performs DFS and assigns SCC values

mark v as **explored**

KOSARAJU'S ALGORITHM

input: a directed graph $G = (V, E)$

output: all SCCs of G

① $G^{\text{rev}} :=$ reverse edges of G
mark all vertices of G^{rev} as **unexplored**

② Compute labeling f via topological ordering algo on G^{rev}

③ mark all vertices of G as **unexplored**
global **numSCC** := 0

for each v in increasing order of $f[v]$

 if v is **unexplored**
 increment **numSCC** by 1
 DFS-SCC(G, v)

DFS-SCC(G, v)

// performs DFS and assigns SCC values

mark v as **explored**

scc(v) := **numSCC**

KOSARAJU'S ALGORITHM

input: a directed graph $G = (V, E)$

output: all SCCs of G

① $G^{\text{rev}} :=$ reverse edges of G
mark all vertices of G^{rev} as **unexplored**

② Compute labeling f via topological ordering algo on G^{rev}

③ mark all vertices of G as **unexplored**
global **numSCC** := 0

for each v in increasing order of $f[v]$

 if v is **unexplored**
 increment **numSCC** by 1
 DFS-SCC(G, v)

DFS-SCC(G, v)

// performs DFS and assigns SCC values

mark v as **explored**

scc(v) := **numSCC**

for each edge (v, w)

 |

KOSARAJU'S ALGORITHM

input: a directed graph $G = (V, E)$

output: all SCCs of G

① $G^{\text{rev}} :=$ reverse edges of G
mark all vertices of G^{rev} as **unexplored**

② Compute labeling f via topological ordering algo on G^{rev}

③ mark all vertices of G as **unexplored**
global **numSCC** := 0

for each v in increasing order of $f[v]$

 if v is **unexplored**
 increment **numSCC** by 1
 DFS-SCC(G, v)

DFS-SCC(G, v)

// performs DFS and assigns SCC values

mark v as **explored**

scc(v) := **numSCC**

for each edge (v, w)

 if w is **unexplored**
 DFS-SCC(G, w)

KOSARAJU'S ALGORITHM

input: a directed graph $G = (V, E)$

output: all SCCs of G

① G^{rev} := reverse edges of G
mark all vertices of G^{rev} as **unexplored**

② Compute labeling f via topological
ordering algo on G^{rev}

③ mark all vertices of G as **unexplored**
global **numSCC** := 0
for each v in increasing order of $f[v]$
| if v is **unexplored**
| | increment **numSCC** by 1
| | DFS-SCC(G, v)

DFS-SCC(G, v)

// performs DFS and assigns
SCC values

mark v as **explored**

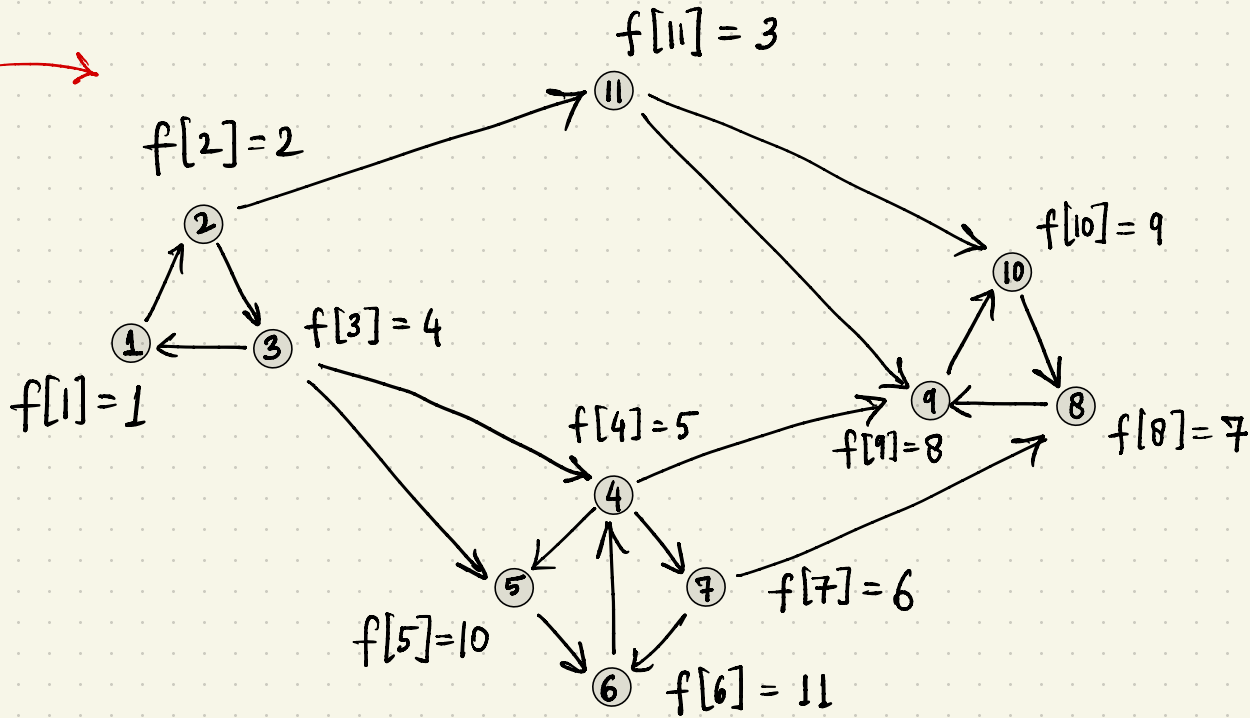
$\text{scc}(v) := \text{numSCC}$

for each edge (v, w)

| if w is **unexplored**
| | DFS-SCC(G, w)

KOSARAJU'S ALGORITHM

G^{rev} $\rightarrow$



KOSARAJU'S ALGORITHM

input: a directed graph $G = (V, E)$

output: all SCCs of G

① $G^{\text{rev}} :=$ reverse edges of G
mark all vertices of G^{rev} as **unexplored**

② Compute labeling f via topological ordering algo on G^{rev}

③ mark all vertices of G as **unexplored**
global **numSCC** := 0

for each v in increasing order of $f[v]$

 if v is **unexplored**
 increment **numSCC** by 1
 DFS-SCC(G, v)

DFS-SCC(G, v)

// performs DFS and assigns SCC values

mark v as **explored**

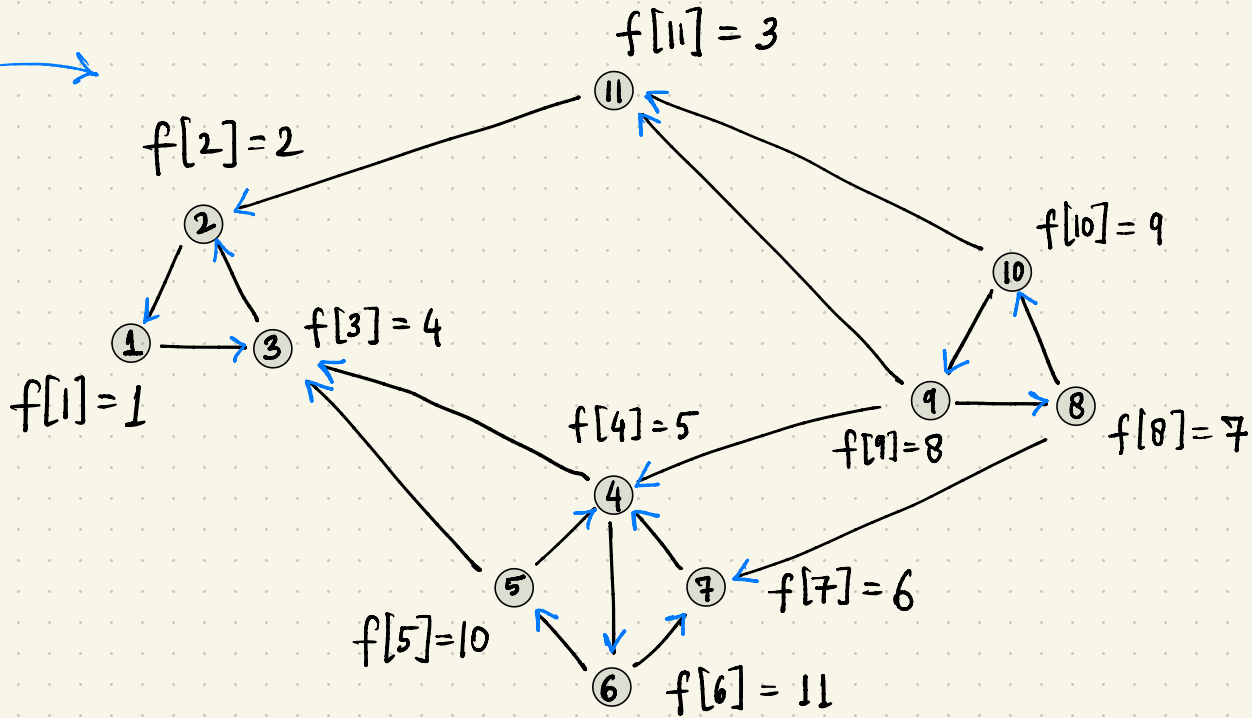
$\text{scc}(v) := \text{numSCC}$

for each edge (v, w)

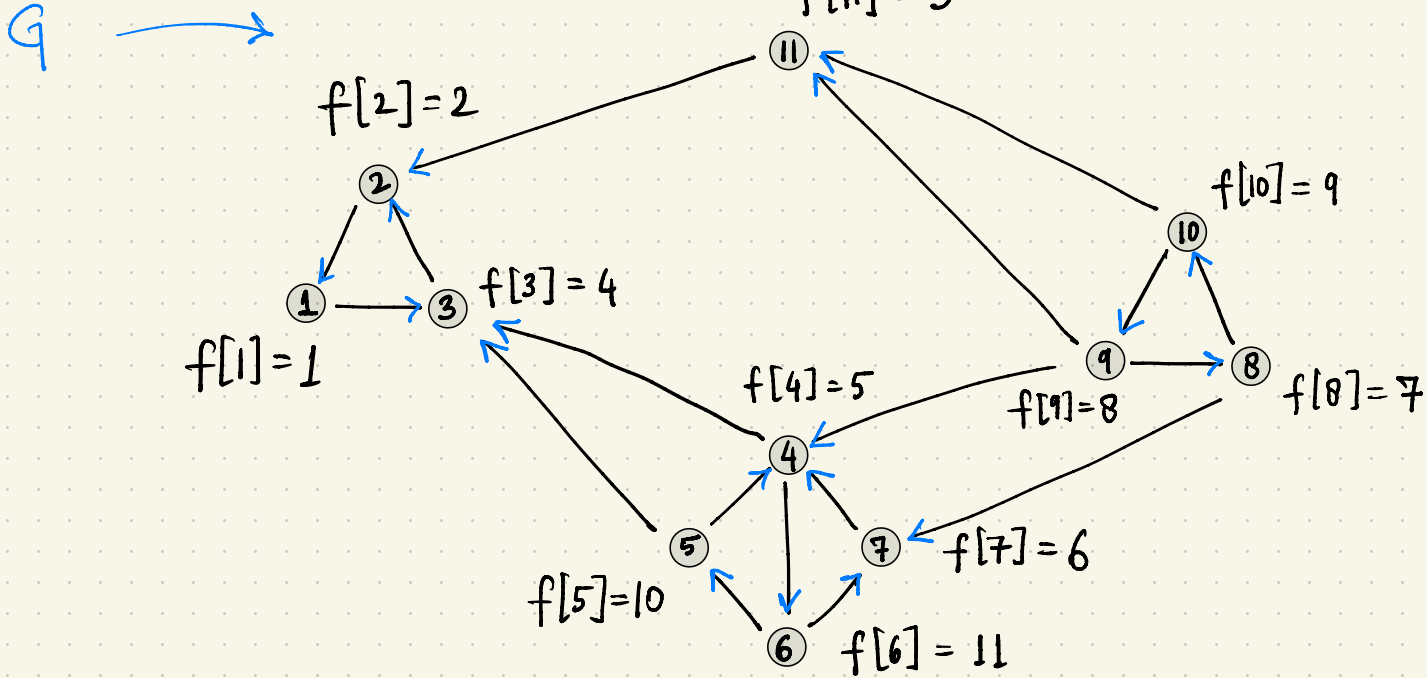
 if w is **unexplored**
 DFS-SCC(G, w)

KOSARAJU'S ALGORITHM

G



KOSARAJU'S ALGORITHM



Recall: f is computed with respect to G^{rev} .

KOSARAJU'S ALGORITHM

input: a directed graph $G = (V, E)$

output: all SCCs of G

① $G^{\text{rev}} :=$ reverse edges of G
mark all vertices of G^{rev} as **unexplored**

② Compute labeling f via topological ordering algo on G^{rev}

③ mark all vertices of G as **unexplored**
global **numSCC** := 0
for each v in **increasing order of $f[v]$**
| if v is **unexplored**
| | increment **numSCC** by 1
| | DFS-SCC(G, v)

DFS-SCC(G, v)

// performs DFS and assigns SCC values

mark v as **explored**

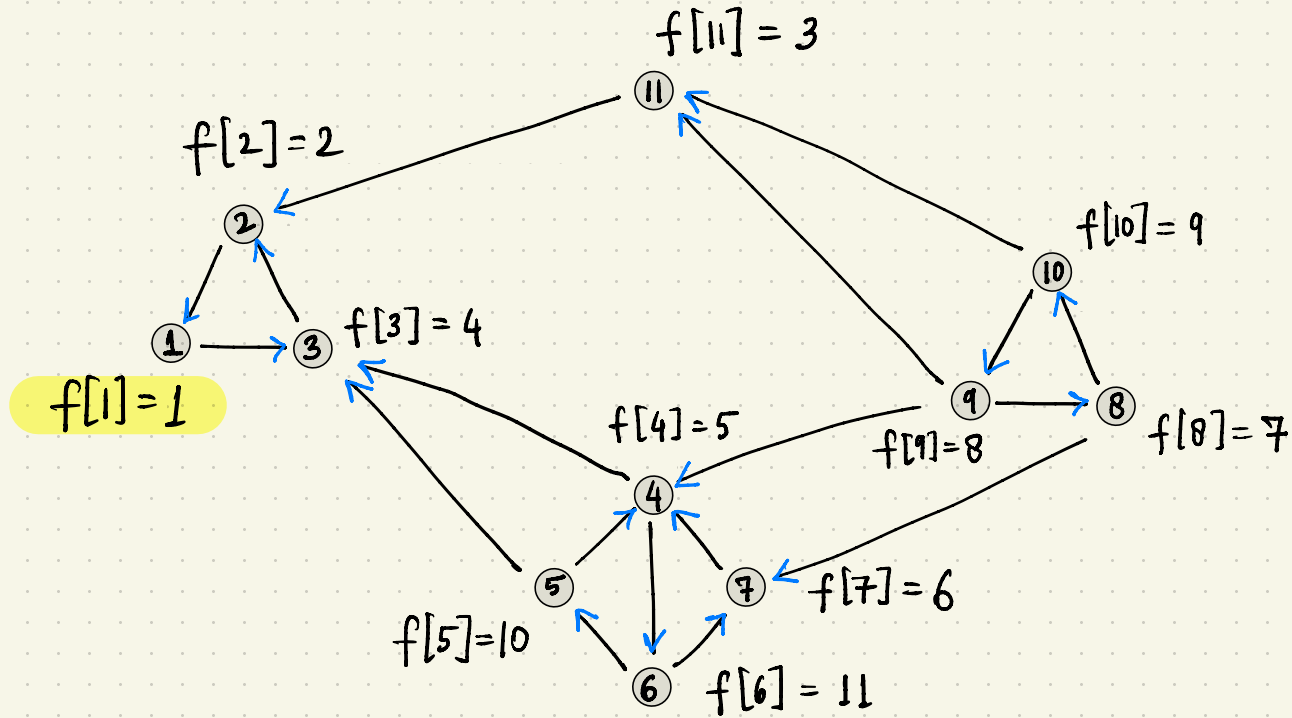
scc(v) := **numSCC**

for each edge (v, w)

| if w is **unexplored**
| | DFS-SCC(G, w)

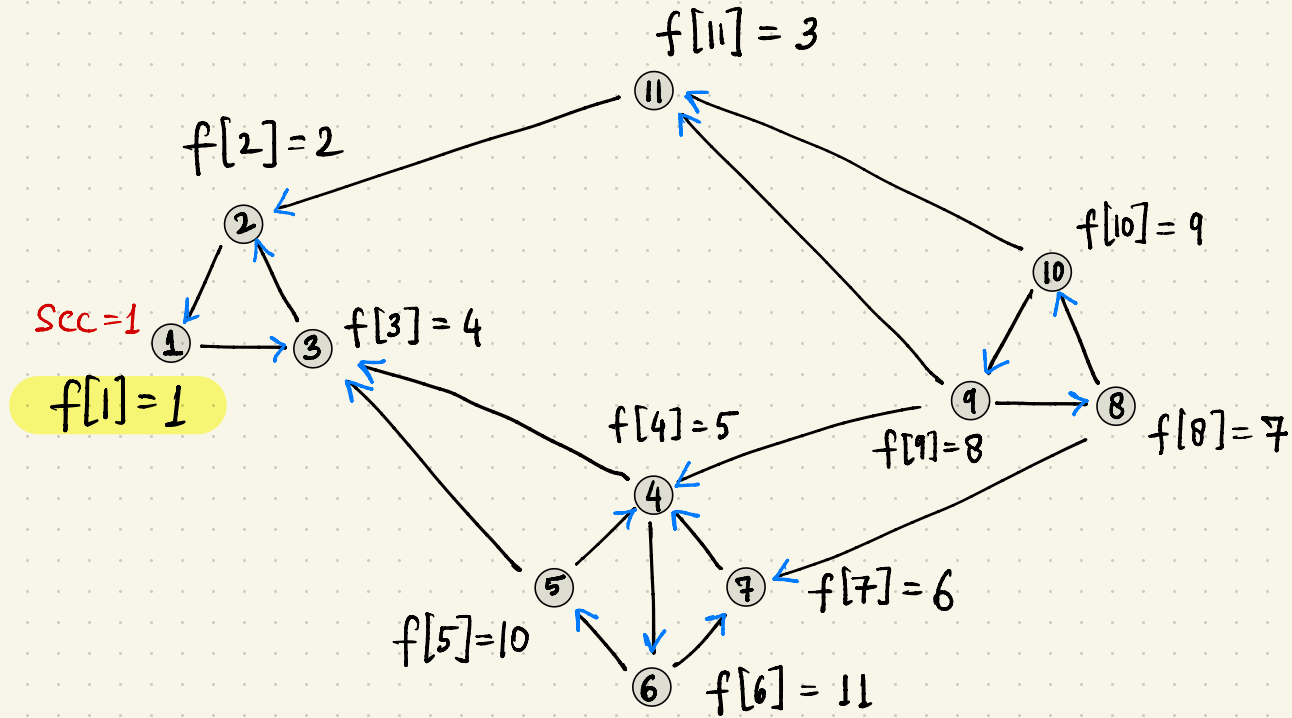
KOSARAJU'S ALGORITHM

9



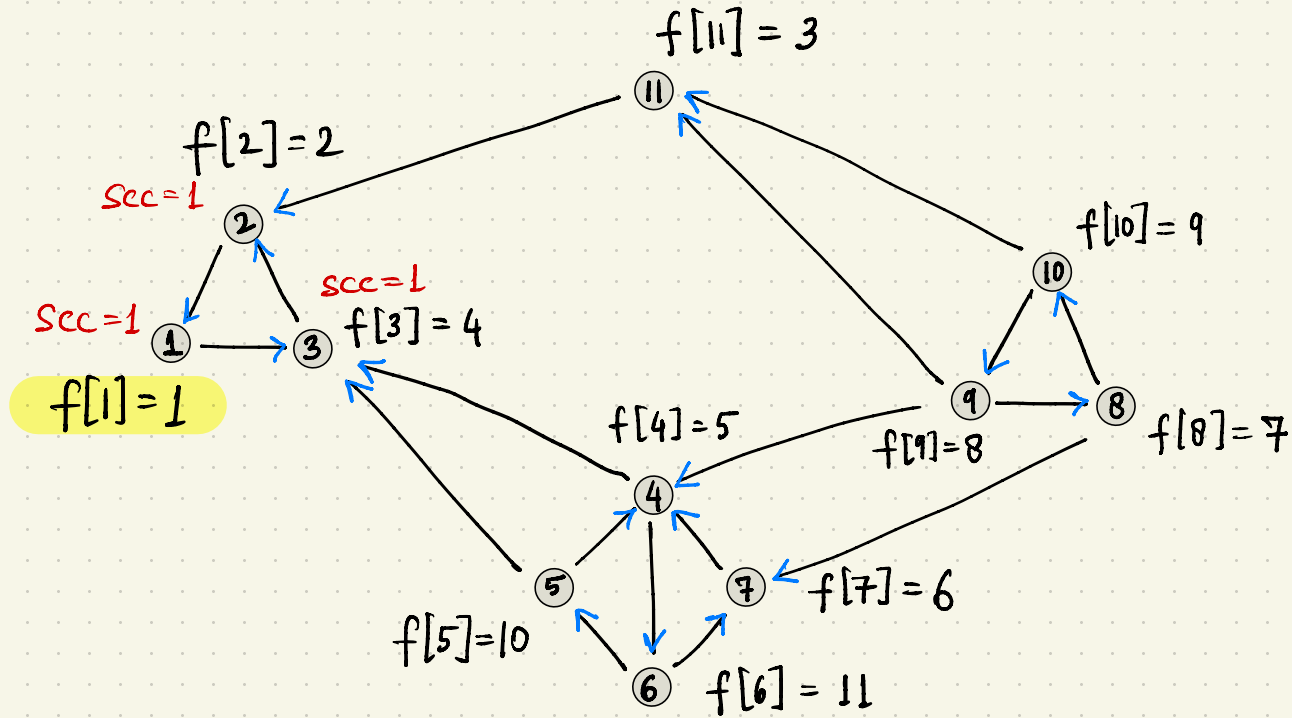
KOSARAJU'S ALGORITHM

9



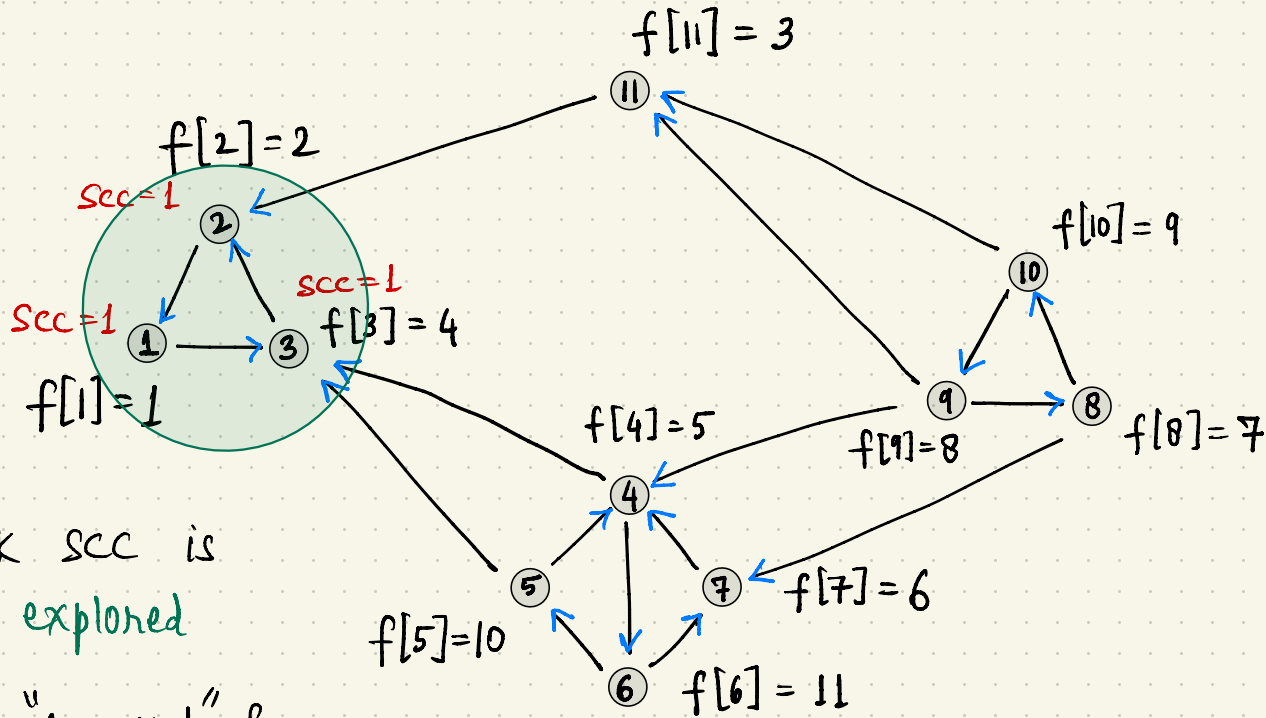
KOSARAJU'S ALGORITHM

9



KOSARAJU'S ALGORITHM

9



The sink SCC is
now explored

(effectively "removed" for
the rest of the algorithm)

KOSARAJU'S ALGORITHM

input: a directed graph $G = (V, E)$

output: all SCCs of G

① $G^{\text{rev}} :=$ reverse edges of G
mark all vertices of G^{rev} as **unexplored**

② Compute labeling f via topological ordering algo on G^{rev}

③ mark all vertices of G as **unexplored**
global **numSCC** := 0
for each v in **increasing order of $f[v]$**
| if v is **unexplored**
| | increment **numSCC** by 1
| | DFS-SCC(G, v)

DFS-SCC(G, v)

// performs DFS and assigns SCC values

mark v as **explored**

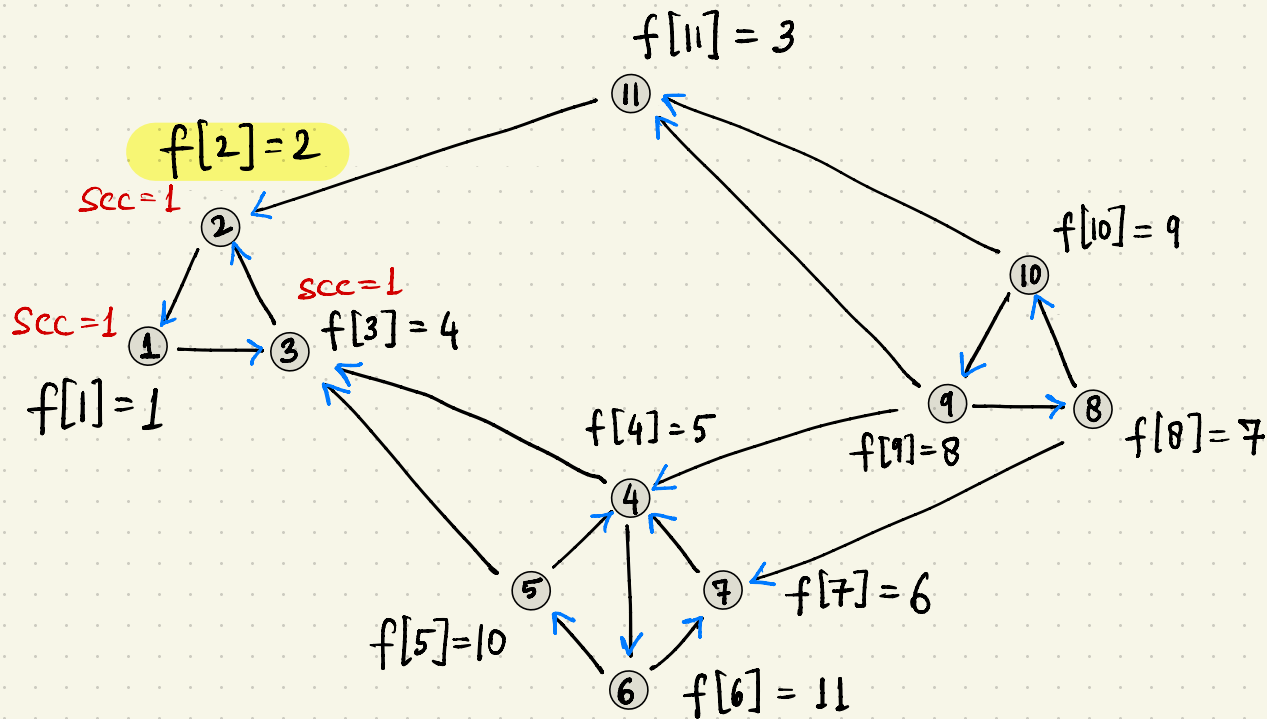
scc(v) := **numSCC**

for each edge (v, w)

| if w is **unexplored**
| | DFS-SCC(G, w)

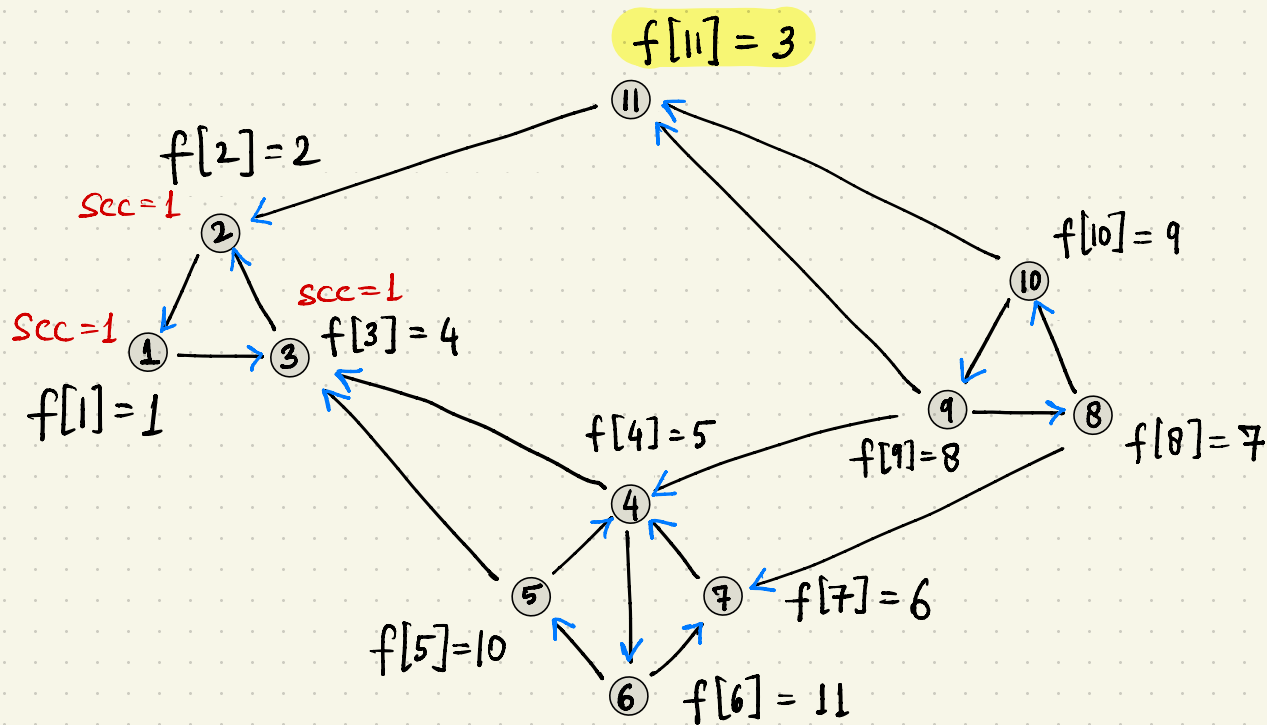
KOSARAJU'S ALGORITHM

9



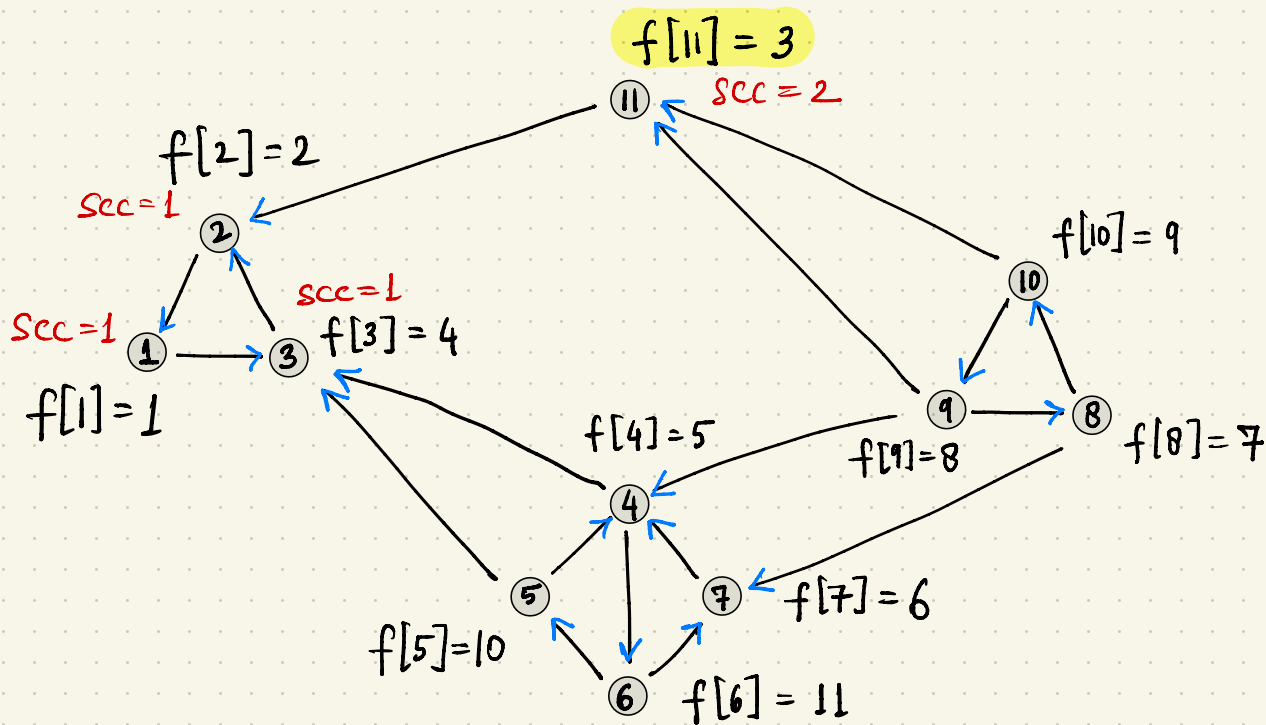
KOSARAJU'S ALGORITHM

9



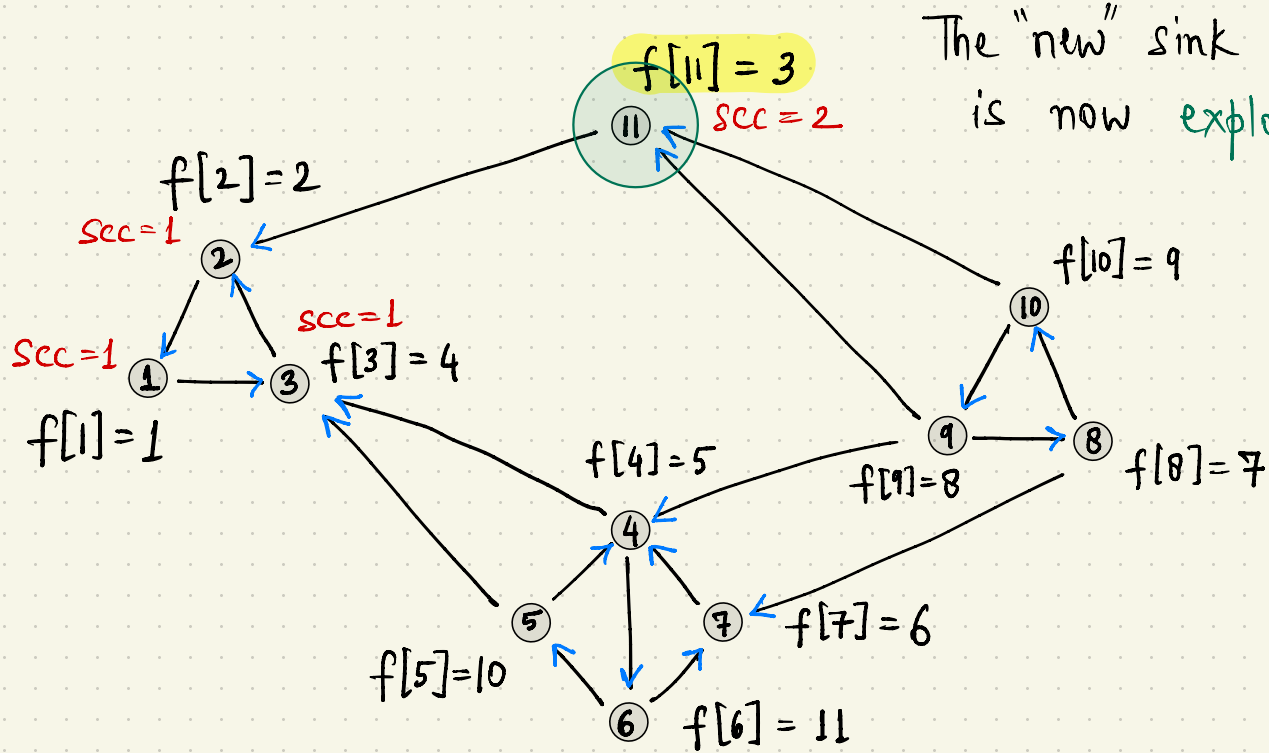
KOSARAJU'S ALGORITHM

9



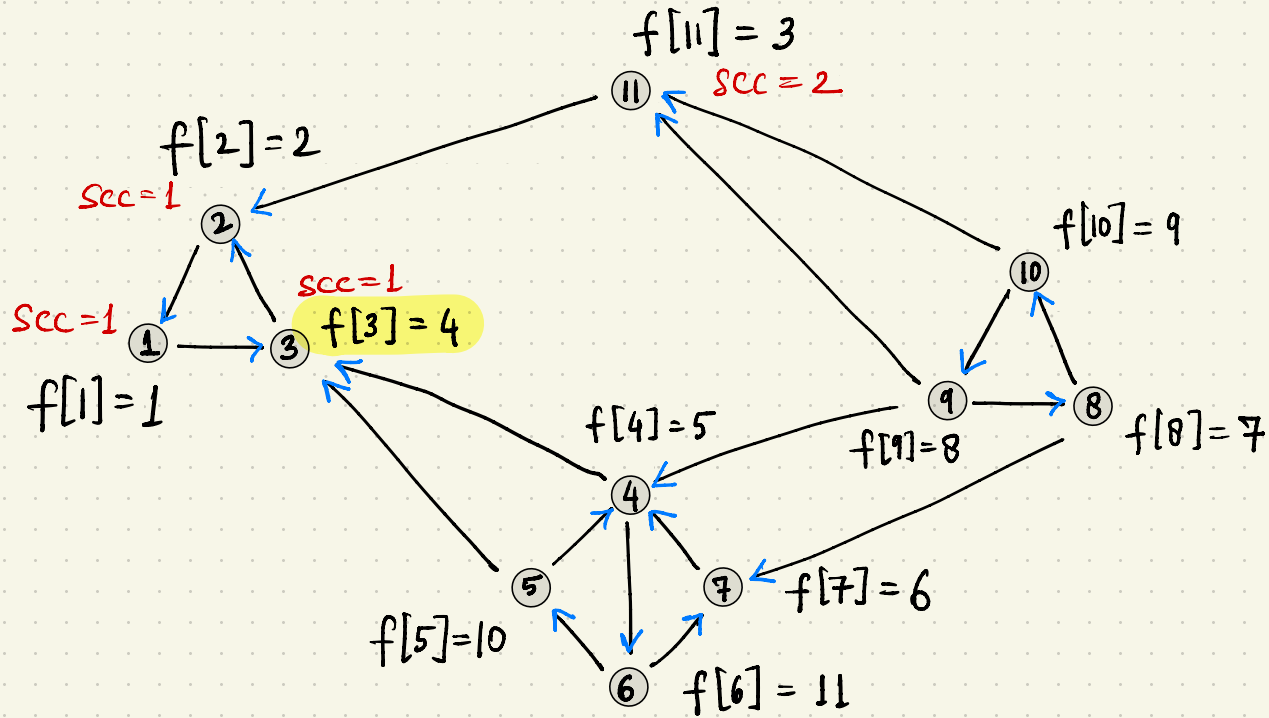
KOSARAJU'S ALGORITHM

9



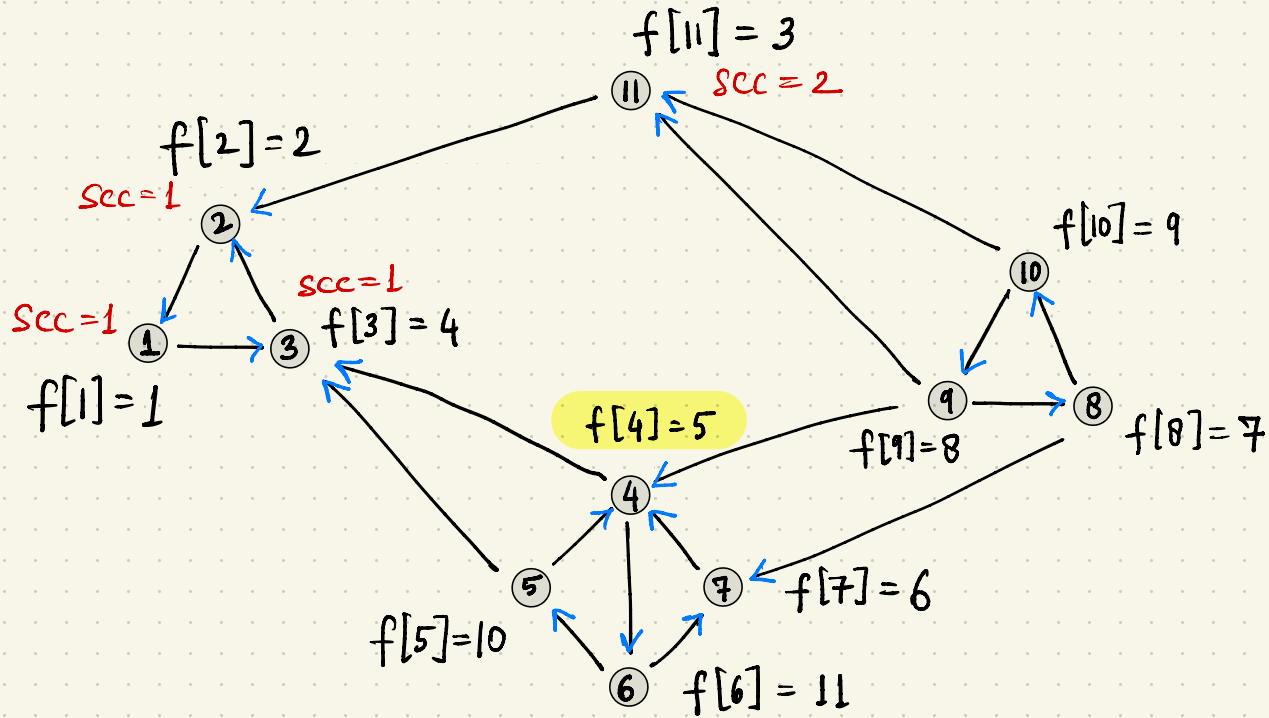
KOSARAJU'S ALGORITHM

9



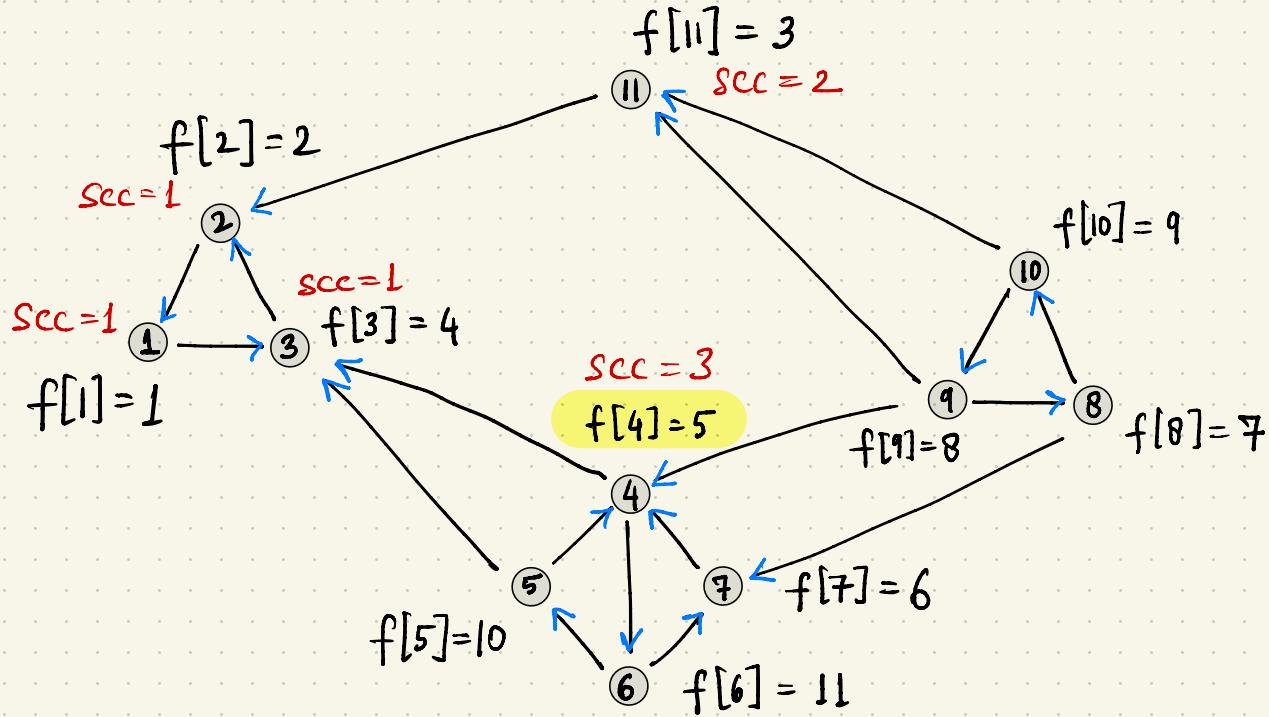
KOSARAJU'S ALGORITHM

9



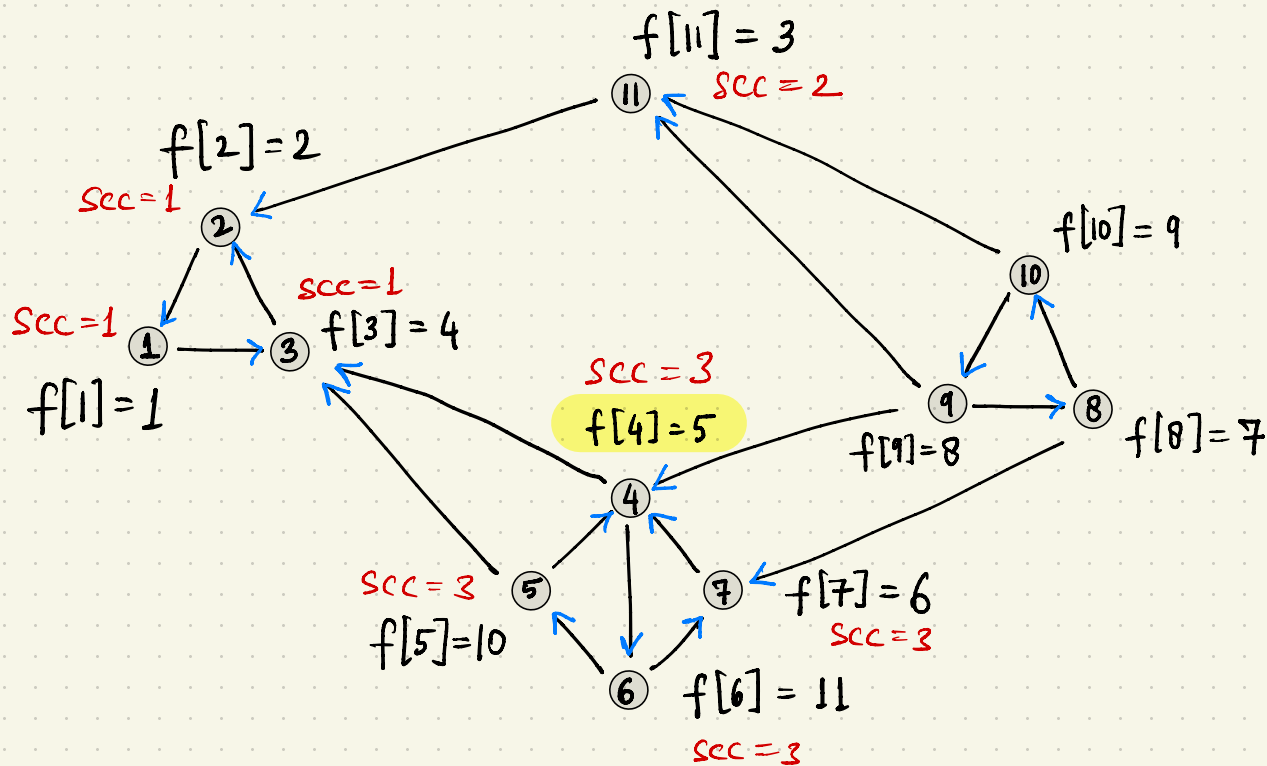
KOSARAJU'S ALGORITHM

9



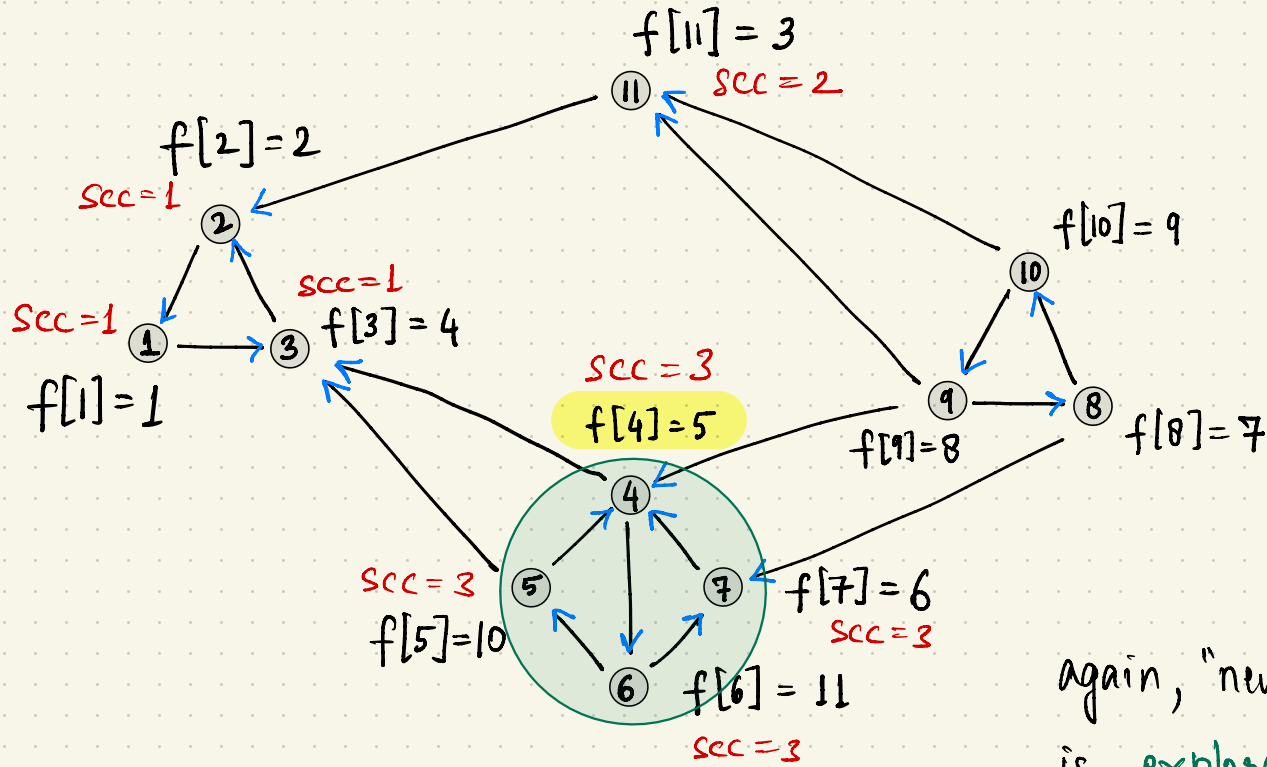
KOSARAJU'S ALGORITHM

9



KOSARAJU'S ALGORITHM

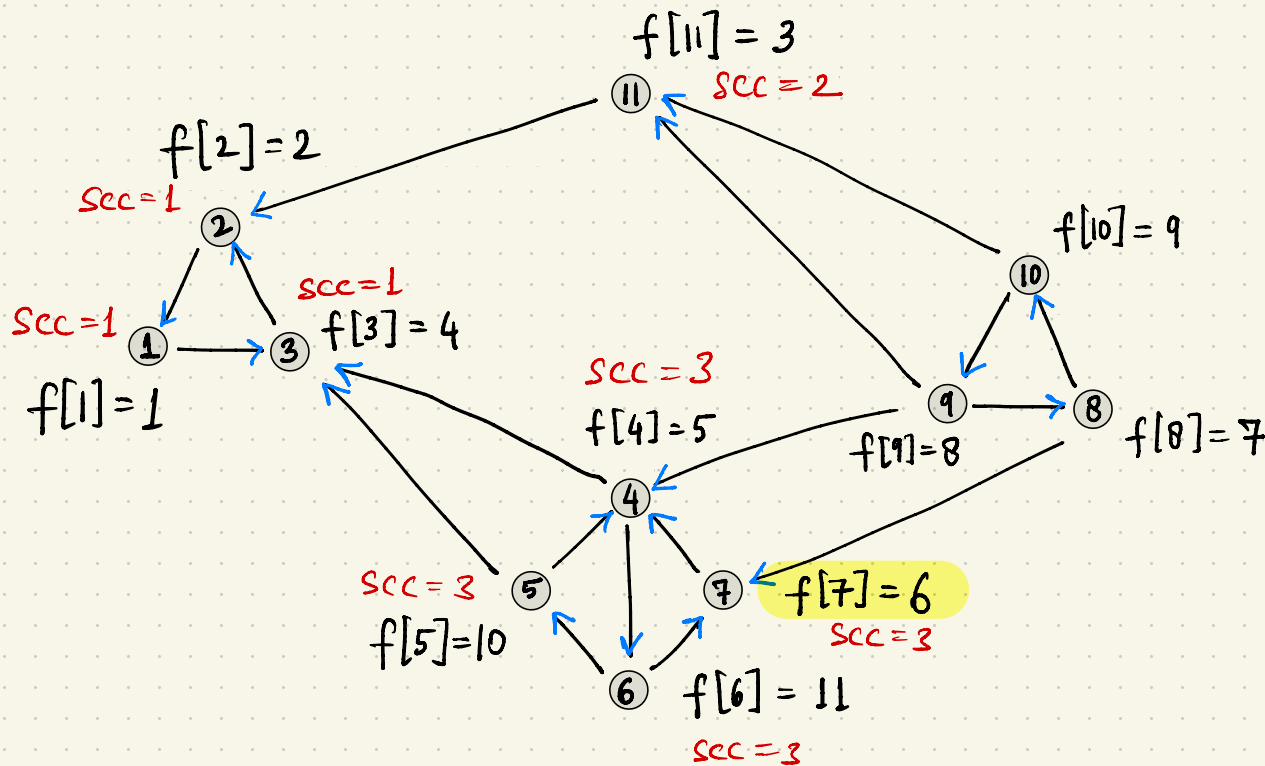
9



again, "new" sink SCC
is explored

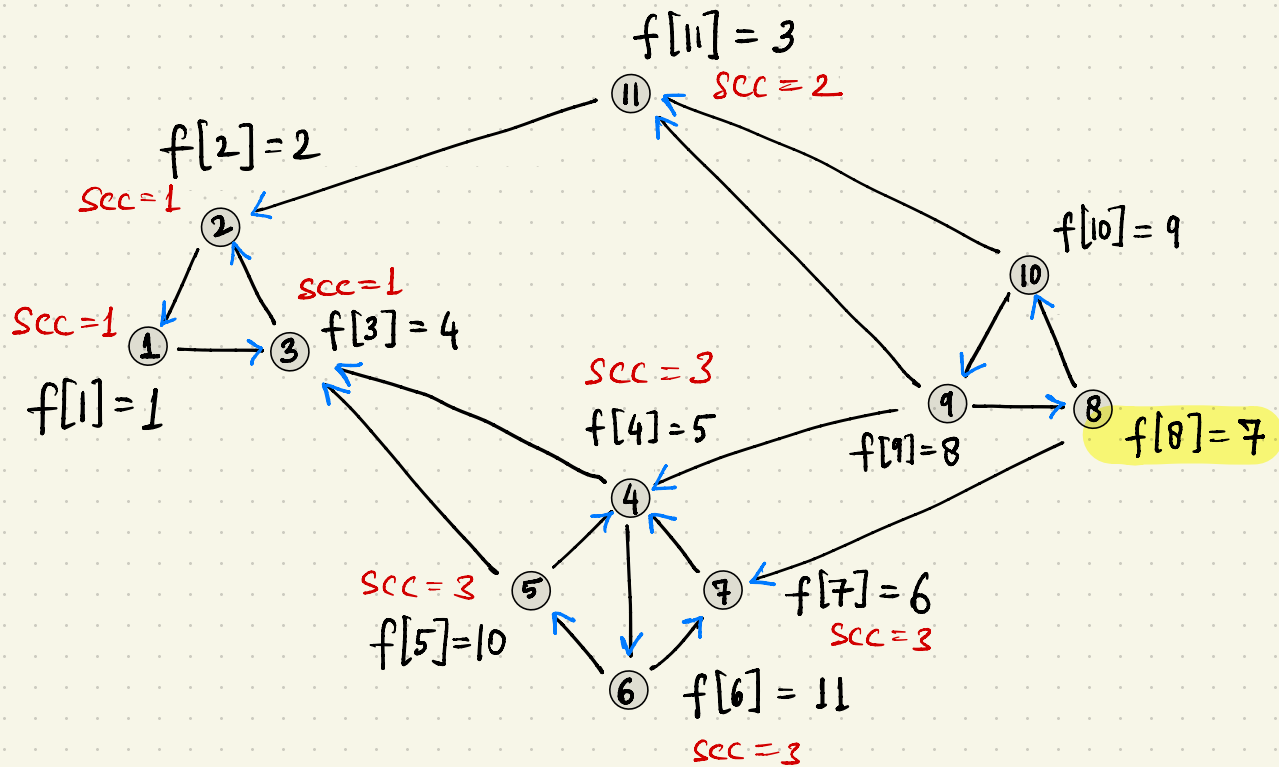
KOSARAJU'S ALGORITHM

9



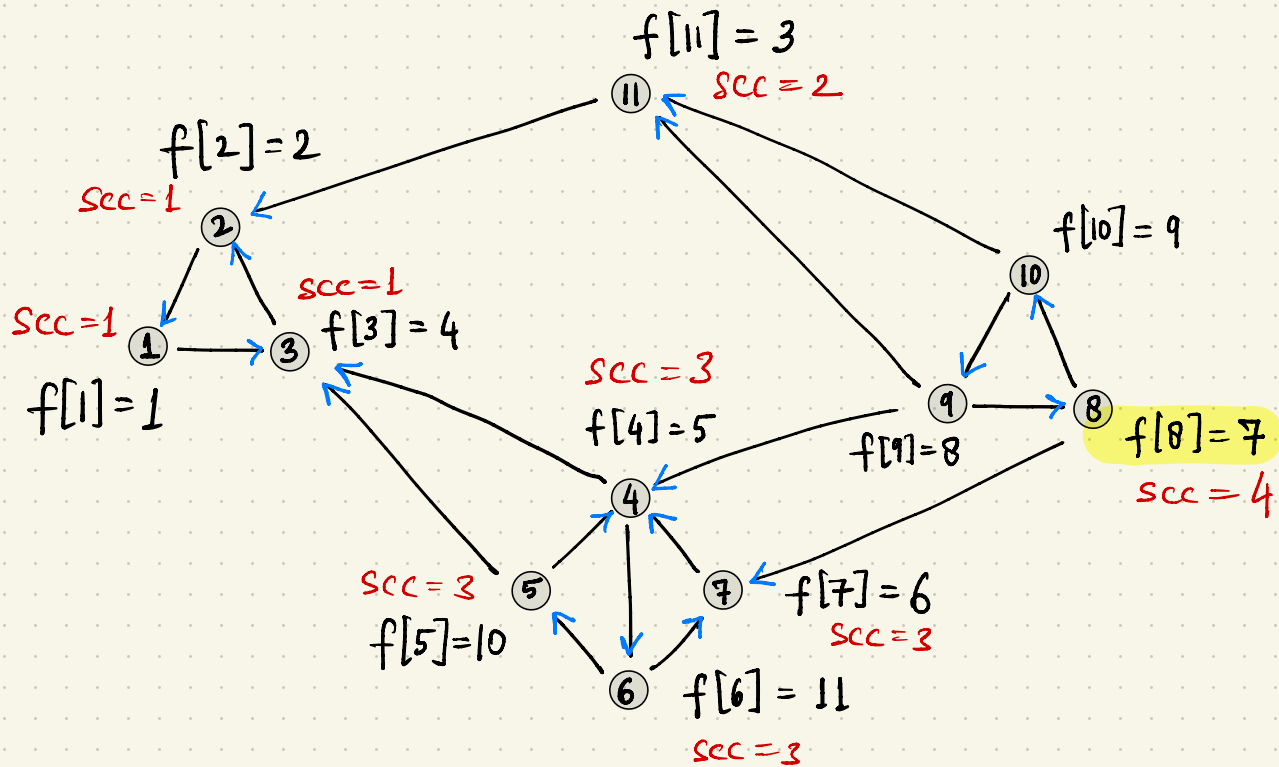
KOSARAJU'S ALGORITHM

9



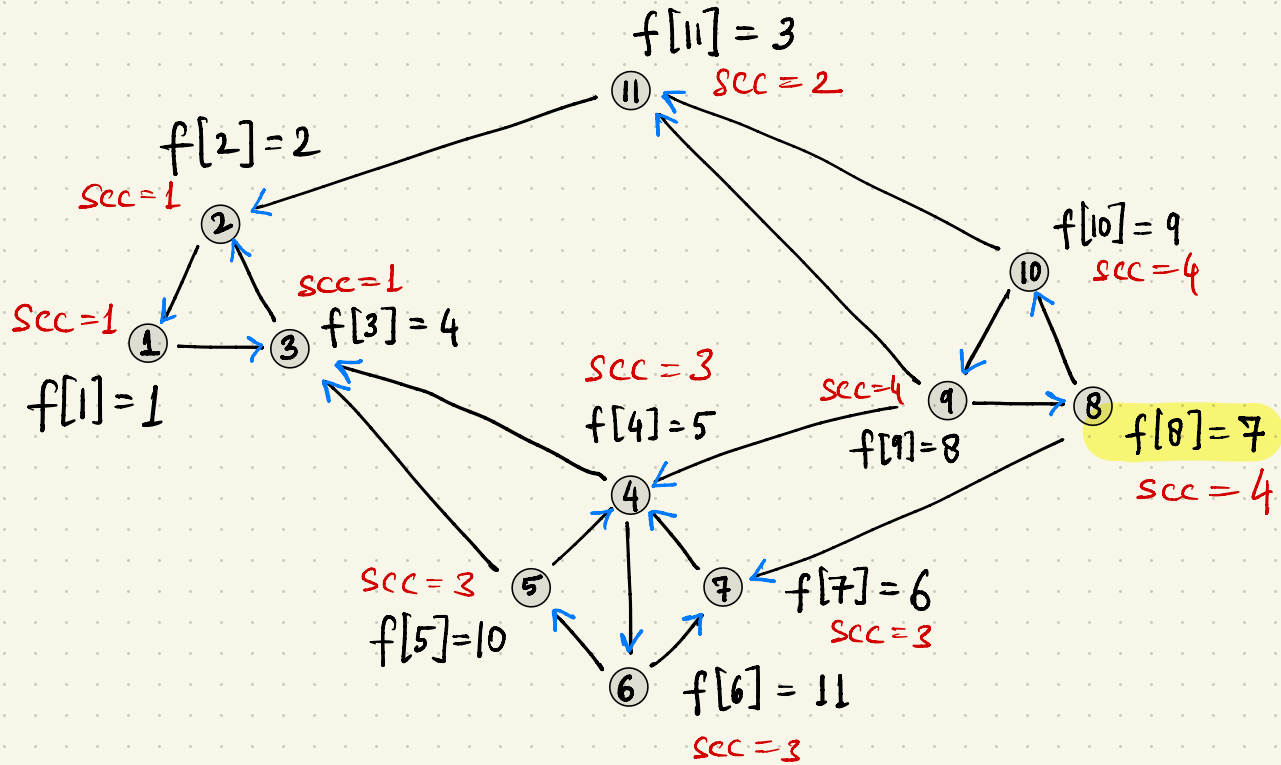
KOSARAJU'S ALGORITHM

9



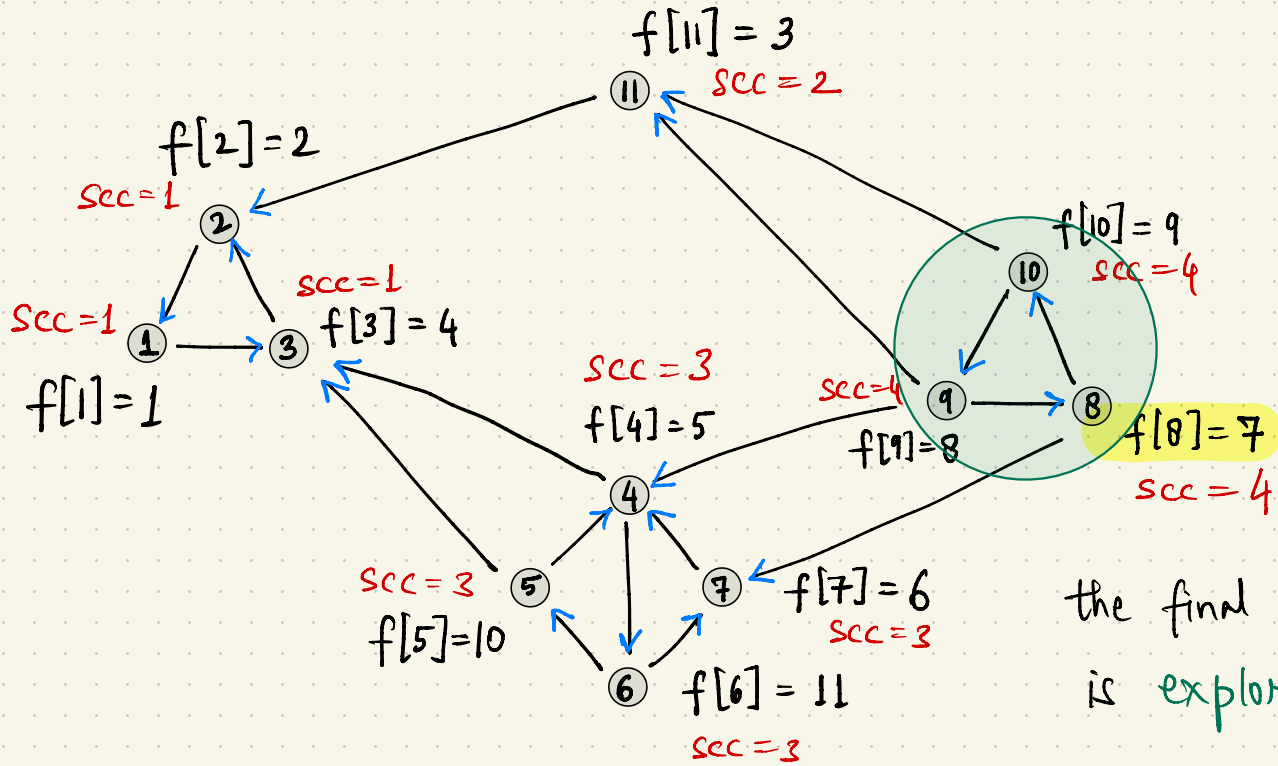
KOSARAJU'S ALGORITHM

9



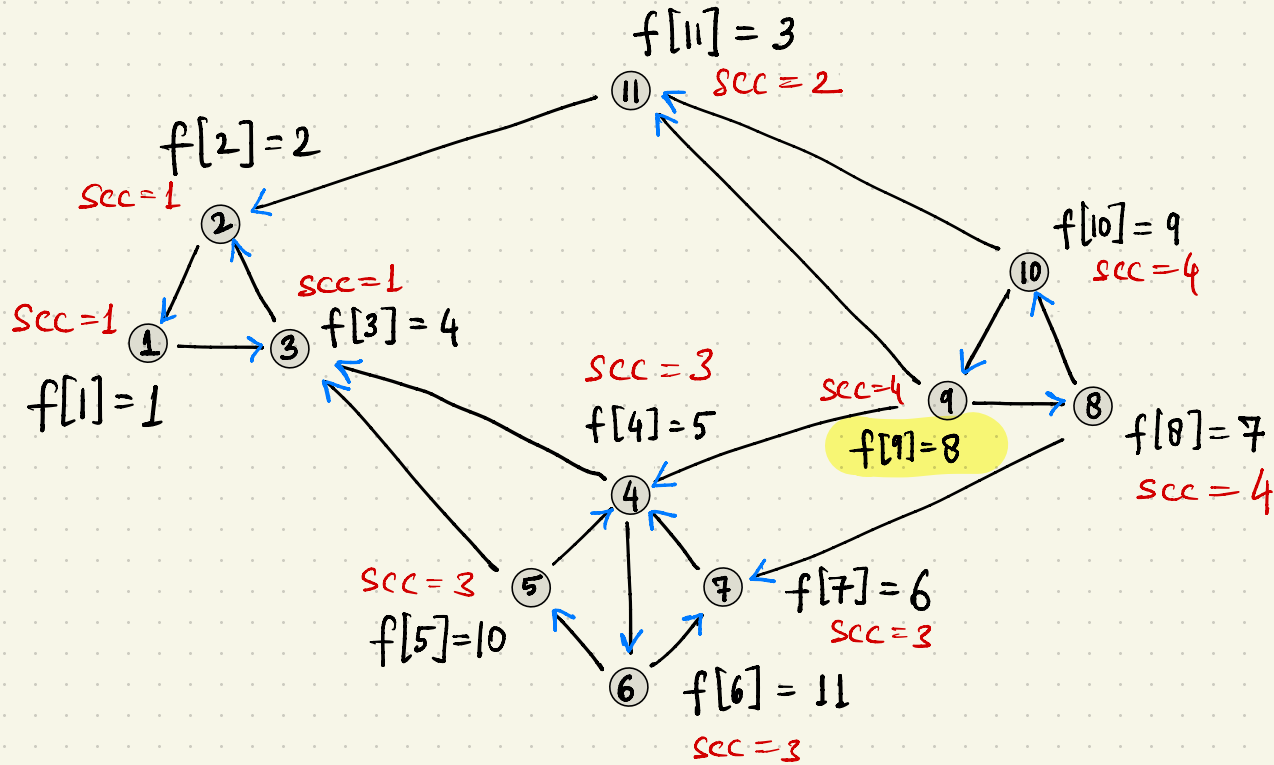
KOSARAJU'S ALGORITHM

9



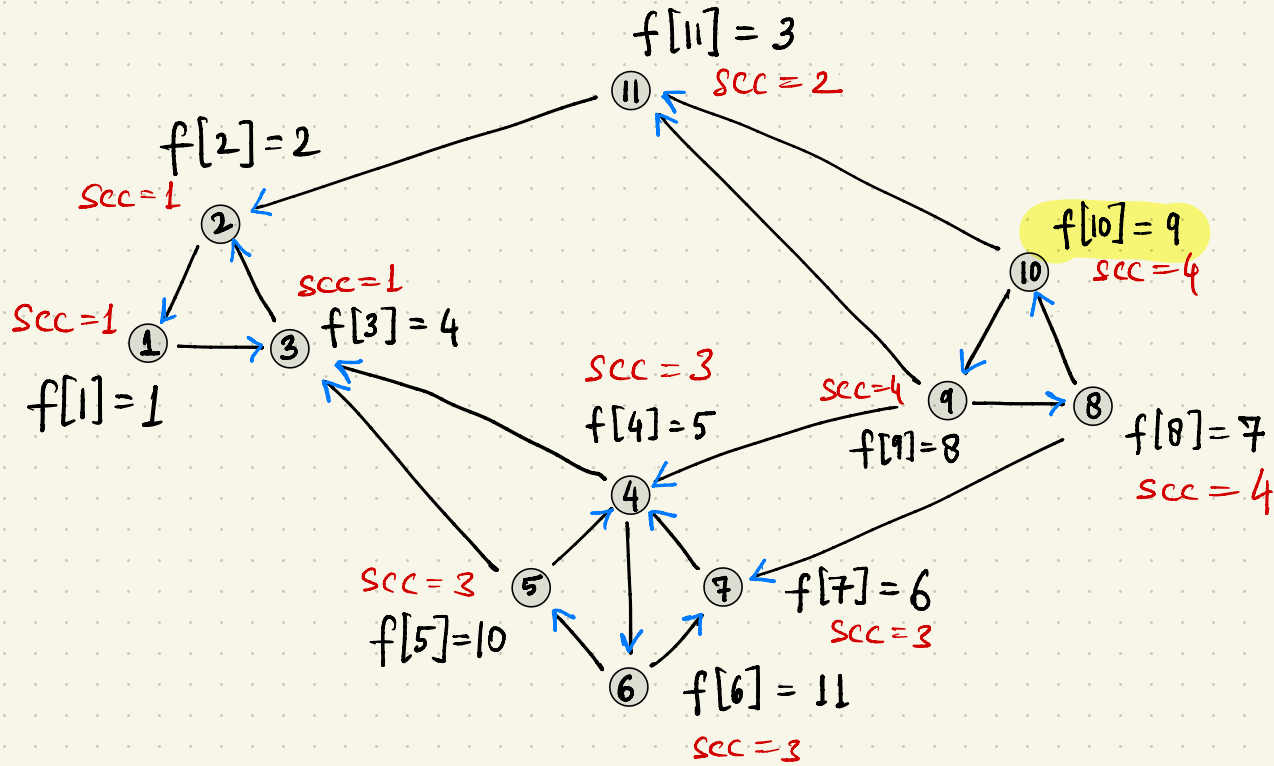
KOSARAJU'S ALGORITHM

9



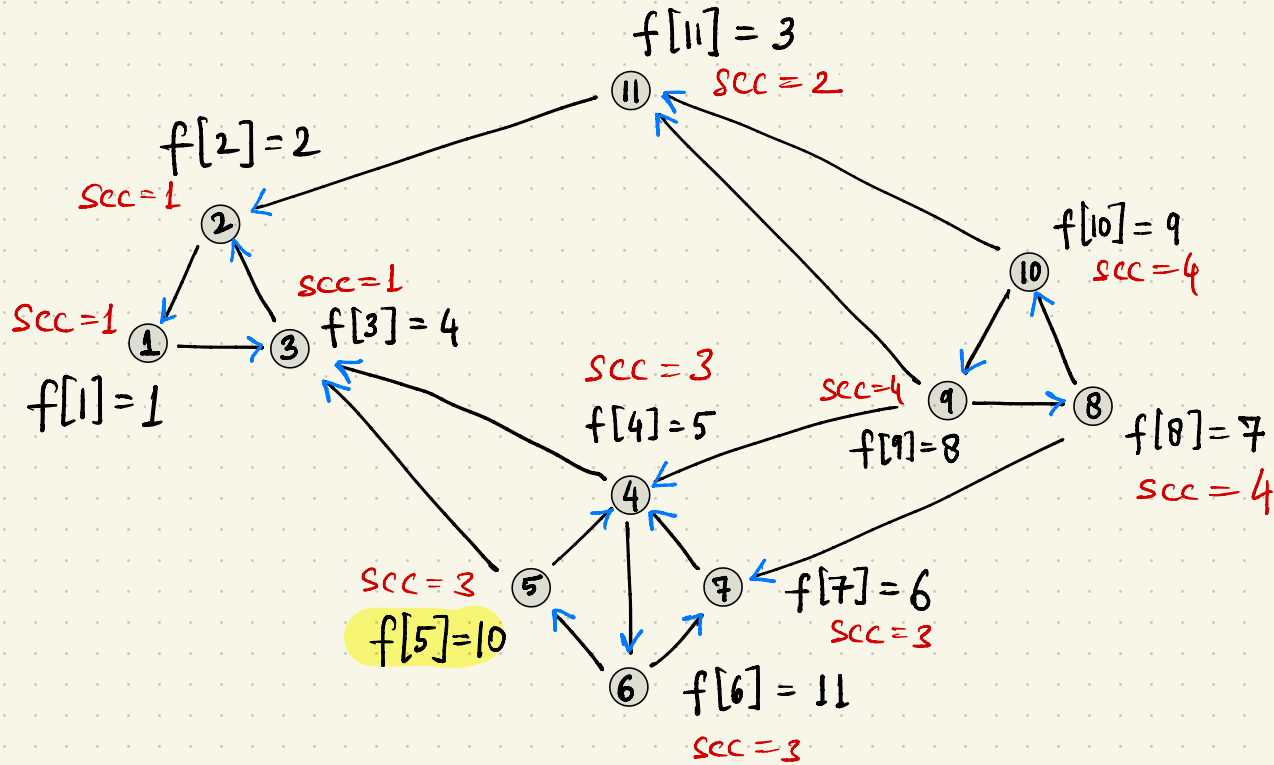
KOSARAJU'S ALGORITHM

9



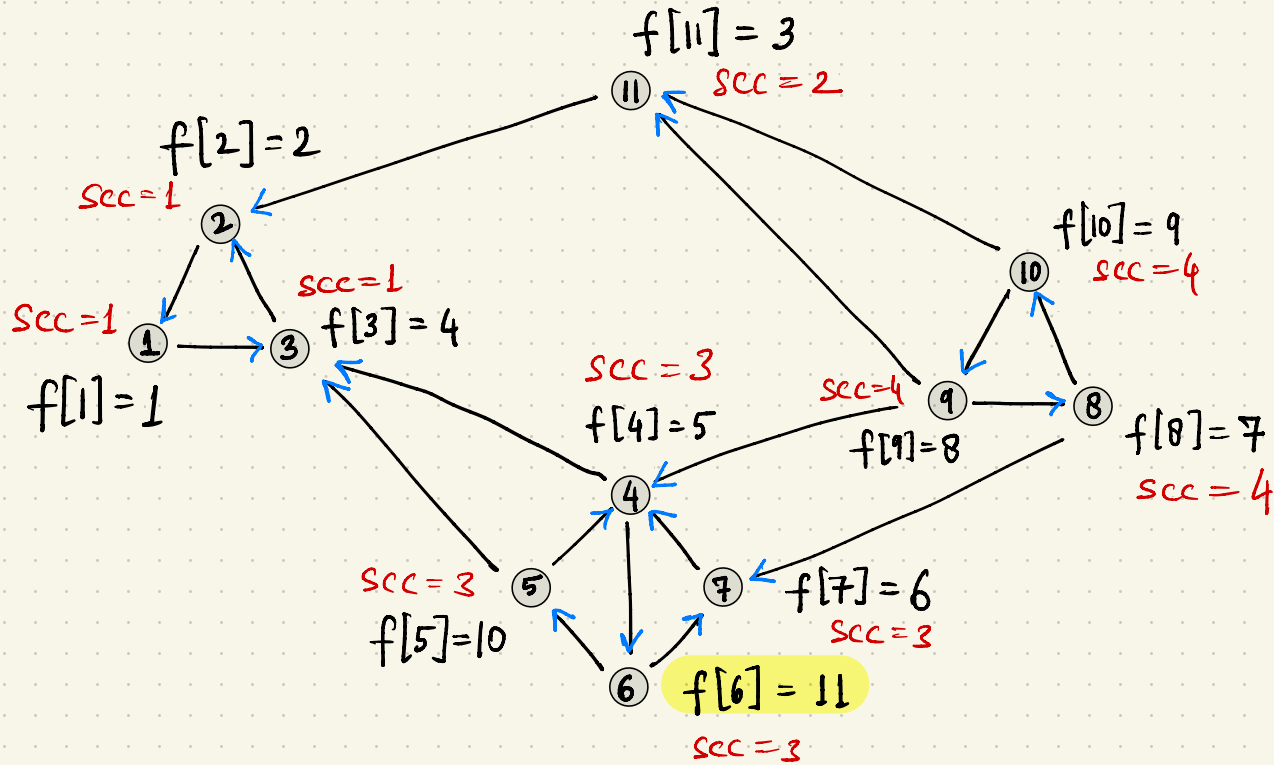
KOSARAJU'S ALGORITHM

9



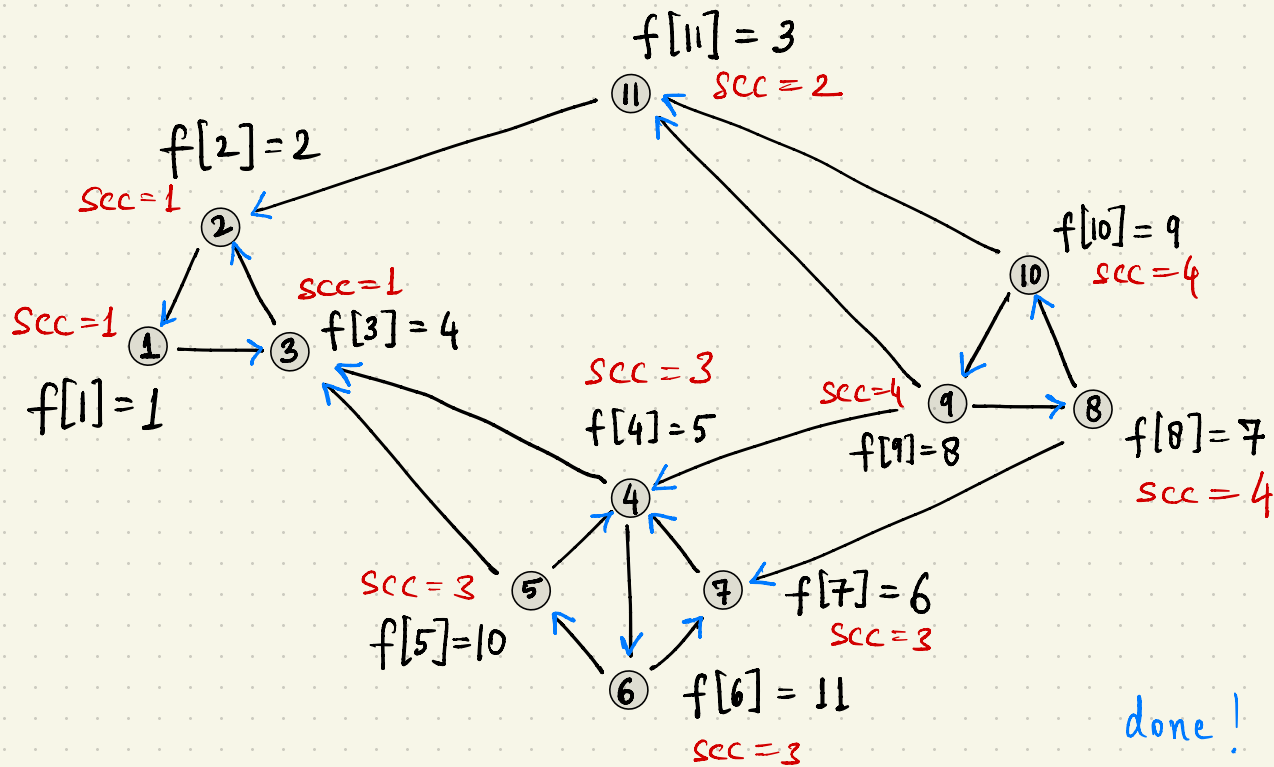
KOSARAJU'S ALGORITHM

9



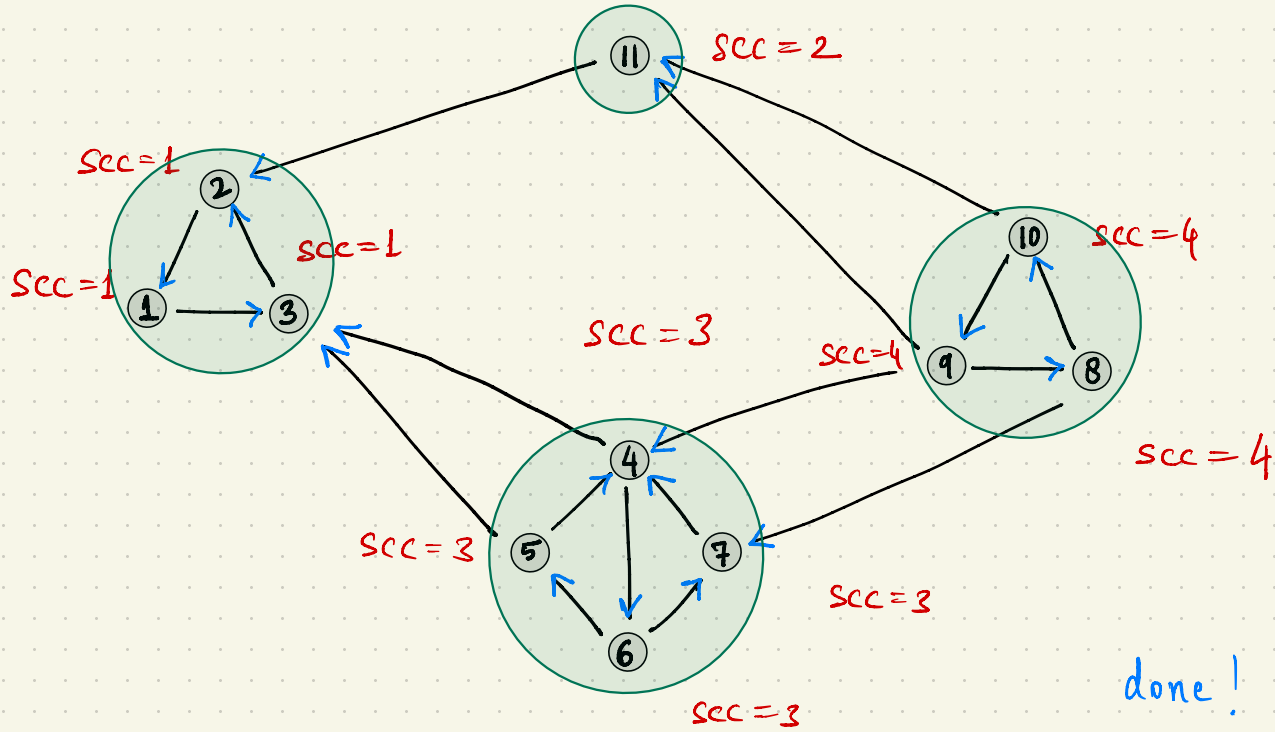
KOSARAJU'S ALGORITHM

G



KOSARAJU'S ALGORITHM

9



KOSARAJU'S ALGORITHM

Theorem: Kosaraju's algorithm terminates in $O(m+n)$ time.

KOSARAJU'S ALGORITHM

Theorem: Kosaraju's algorithm terminates in $O(m+n)$ time.

Proof: Exercise

Idea: Two runs of DFS.

KOSARAJU'S ALGORITHM

Theorem: Kosaraju's algorithm terminates in $O(m+n)$ time.

Proof: Exercise

Idea: Two runs of DFS.

🤔 To go through $f[v]$ in ascending order, do we need to sort?

KOSARAJU'S ALGORITHM

Theorem: At termination of Kosaraju's algorithm, for any $u, v \in V$

$scc[u] = scc[v] \iff u$ and v are in the same
strongly connected component

KOSARAJU'S ALGORITHM

Theorem: At termination of Kosaraju's algorithm, for any $u, v \in V$

$scc[u] = scc[v] \iff u$ and v are in the same
strongly connected component

Proof: ($\Leftarrow$)

KOSARAJU'S ALGORITHM

Theorem: At termination of Kosaraju's algorithm, for any $u, v \in V$

$scc[u] = scc[v] \iff u$ and v are in the same
strongly connected component

Proof: ($\Leftarrow$) Suppose u and v in the same SCC, say S .

KOSARAJU'S ALGORITHM

Theorem: At termination of Kosaraju's algorithm, for any $u, v \in V$

$scc[u] = scc[v] \iff u$ and v are in the same strongly connected component

Proof: ($\Leftarrow$) Suppose u and v in the same SCC, say S .

Let $x \in S$ be the first explored vertex in S ("leader" vertex)

KOSARAJU'S ALGORITHM

Theorem: At termination of Kosaraju's algorithm, for any $u, v \in V$

$\text{sec}[u] = \text{sec}[v] \iff u \text{ and } v \text{ are in the same strongly connected component}$

Proof: ($\Leftarrow$) Suppose u and v in the same SCC, say S .

Let $x \in S$ be the first explored vertex in S ("leader" vertex)

By correctness of DFS, $\text{sec}[u] = \text{sec}[v] = \text{sec}[x]$.

KOSARAJU'S ALGORITHM

Theorem: At termination of Kosaraju's algorithm, for any $u, v \in V$

$scc[u] = scc[v] \iff u$ and v are in the same
strongly connected component

Proof: ($\Rightarrow$)

KOSARAJU'S ALGORITHM

Theorem: At termination of Kosaraju's algorithm, for any $u, v \in V$

$scc[u] = scc[v] \iff u$ and v are in the same strongly connected component

Proof: $(\implies)$ (by strong induction)

$P(i) : scc[u] = scc[v] = i \implies u, v$ in same SCC

KOSARAJU'S ALGORITHM

Theorem: At termination of Kosaraju's algorithm, for any $u, v \in V$

$scc[u] = scc[v] \iff u$ and v are in the same strongly connected component

Proof: $(\implies)$ (by strong induction)

$P(i) : scc[u] = scc[v] = i \implies u, v$ in same SCC

Base case $P(1) : scc[u] = scc[v] = 1$

$\implies$ must belong to sink SCC of G
(by key observation).

KOSARAJU'S ALGORITHM

Theorem: At termination of Kosaraju's algorithm, for any $u, v \in V$

$scc[u] = scc[v] \iff u$ and v are in the same
strongly connected component

Proof: ($\Rightarrow$) (by strong induction)

$P(i+1) : scc[u] = scc[v] = i+1$

KOSARAJU'S ALGORITHM

Theorem: At termination of Kosaraju's algorithm, for any $u, v \in V$

$scc[u] = scc[v] \iff u$ and v are in the same
strongly connected component

Proof: $(\implies)$ (by strong induction)

$P(i+1) : scc[u] = scc[v] = i+1$

Among vertices with $scc[\cdot] = i+1$, let u be the first
to be marked as **explored**.

KOSARAJU'S ALGORITHM

Theorem: At termination of Kosaraju's algorithm, for any $u, v \in V$

$scc[u] = scc[v] \iff u$ and v are in the same strongly connected component

Proof: ($\Rightarrow$) (by strong induction)

$P(i+1) : scc[u] = scc[v] = i+1$

Then, all vertices in SCC of π get $scc[\cdot] = i+1$.

KOSARAJU'S ALGORITHM

Theorem: At termination of Kosaraju's algorithm, for any $u, v \in V$

$scc[u] = scc[v] \iff u$ and v are in the same strongly connected component

Proof: ($\Rightarrow$) (by strong induction)

$P(i+1) : scc[u] = scc[v] = i+1$

Then, all vertices in SCC of π get $scc[\cdot] = i+1$.

No other vertex gets $scc[\cdot] = i+1$. **Why?**

KOSARAJU'S ALGORITHM

Theorem: At termination of Kosaraju's algorithm, for any $u, v \in V$

$scc[u] = scc[v] \iff u$ and v are in the same strongly connected component

Proof: ($\Rightarrow$) (by strong induction)

$P(i+1) : scc[u] = scc[v] = i+1$

Then, all vertices in SCC of π get $scc[\cdot] = i+1$.

No other vertex gets $scc[\cdot] = i+1$. **Why?**

SCC $i+1$ can only have edges to SCC $1, 2, \dots, i$ — explored!

