Lecture 14: Basic Graph Algorithms II

Michael Dinitz

October 10, 2024

601.433/633 Introduction to Algorithms

Introduction

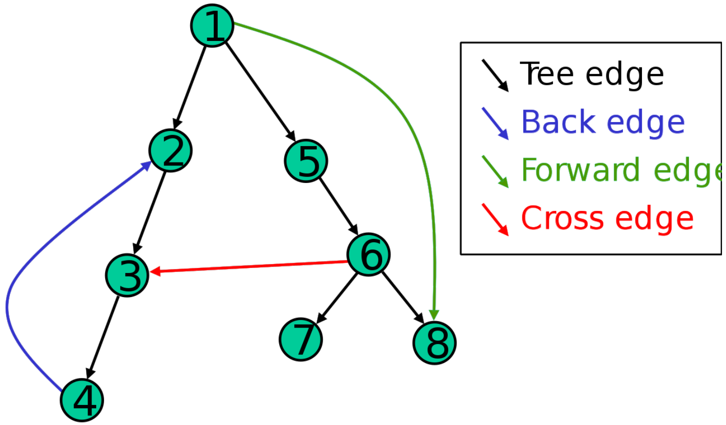
Last time: BFS and DFS

Today: Topological Sort, Strongly Connected Components

- Both very classical and important uses of DFS!

Edge Types

DFS naturally gives a spanning forest: edge (v, u) if DFS(v) calls DFS(u)



Forward Edges: (v, u) such that u descendent of v (includes tree edges)

$start(v) < start(u) < finish(u) < finish(v)$

Back Edges: (v, u) such that u an ancestor of v

$start(u) < start(v) < finish(v) < finish(u)$

Cross Edges: (v, u) such that u neither a descendent nor an ancestor of v

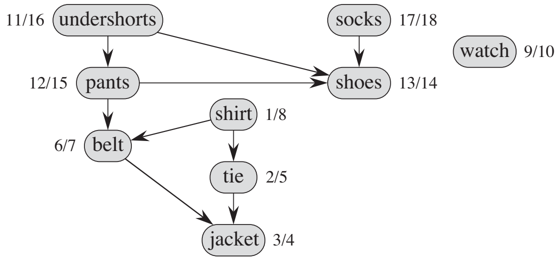
$start(u) < finish(u) < start(v) < finish(v)$

Topological Sort

Definitions

Definition

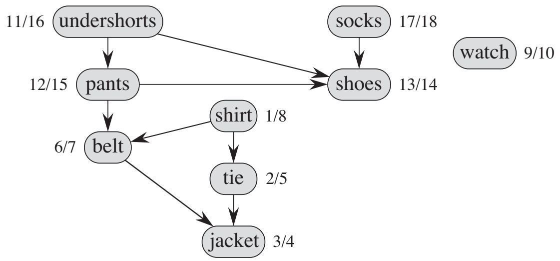
A directed graph G is a *Directed Acyclic Graph (DAG)* if it has no directed cycles.



Definitions

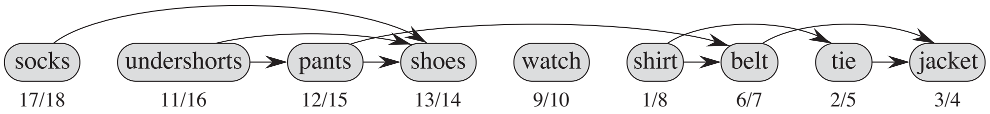
Definition

A directed graph G is a *Directed Acyclic Graph (DAG)* if it has no directed cycles.



Definition

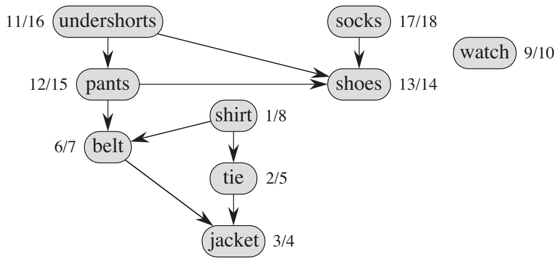
A *topological sort* $v_1, v_2, \ldots, v_n$ of a DAG is an ordering of the vertices such that all edges are of the form (v_i, v_j) with $i < j$.



Definitions

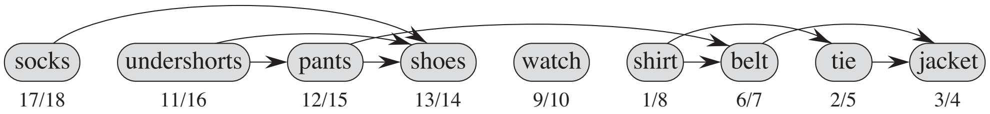
Definition

A directed graph G is a *Directed Acyclic Graph (DAG)* if it has no directed cycles.



Definition

A *topological sort* $v_1, v_2, \ldots, v_n$ of a DAG is an ordering of the vertices such that all edges are of the form (v_i, v_j) with $i < j$.



Q: Can we always topological sort a DAG? How fast?

Topological Sort

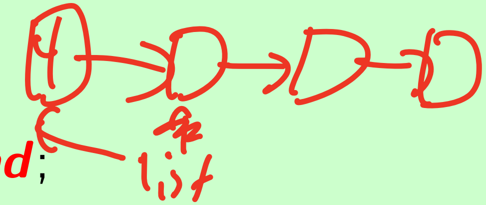
Algorithm (informal): Run DFS(G). When DFS(v) returns, put v at beginning of list

Topological Sort

Algorithm (informal): Run DFS(G). When DFS(v) returns, put v at beginning of list

```
DFS(G) {
    list → head = NULL;
    t = 0;
    for all v ∈ V {
        start(v) = 0;
        finish(v) = 0;
    }
    while ∃v ∈ V with start(v) = 0 {
        DFS(v);
    }
}
```

```
DFS(v) {
    t = t + 1;
    start(v) = t;
    for each edge (v, u) ∈ A[v] {
        if start(u) == 0 then DFS(u);
    }
    t = t + 1;
    finish(v) = t;
    temp = list → head;
    list → head = v;
    list → head → next = temp;
}
```



Characterizing DAGs

Theorem

A directed graph $\boldsymbol{G}$ *is a DAG if and only if DFS*($\boldsymbol{G}$) *has no back edges.*

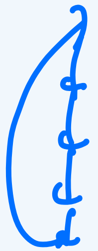
Characterizing DAGs

Theorem

A directed graph $\boldsymbol{G}$ is a DAG if and only if DFS($\boldsymbol{G}$) has no back edges.

Proof.

Only if ($\Rightarrow$): contrapositive. If G has a back edge:



Characterizing DAGs

Theorem

A directed graph $\boldsymbol{G}$ is a DAG if and only if DFS($\boldsymbol{G}$) has no back edges.

Proof.

Only if ($\Rightarrow$): contrapositive. If $\boldsymbol{G}$ has a back edge: Directed cycle! Not a DAG.

Characterizing DAGs

Theorem

A directed graph $\boldsymbol{G}$ is a DAG if and only if DFS($\boldsymbol{G}$) has no back edges.

Proof.

Only if ($\Rightarrow$): contrapositive. If $\boldsymbol{G}$ has a back edge: Directed cycle! Not a DAG.

If ($\Leftarrow$): contrapositive. If $\boldsymbol{G}$ has a directed cycle $\boldsymbol{C}$:

Characterizing DAGs

Theorem

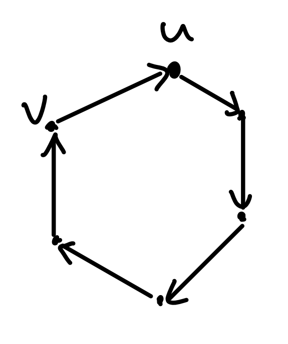
A directed graph G is a DAG if and only if DFS(G) has no back edges.

Proof.

Only if ($\Rightarrow$): contrapositive. If G has a back edge: Directed cycle! Not a DAG.

If ($\Leftarrow$): contrapositive. If G has a directed cycle C:

- Let $u \in C$ with minimum start value, v predecessor in cycle
- All nodes in C reachable from $u \implies$ all nodes in C descendants of u
- (v, u) a back edge



□

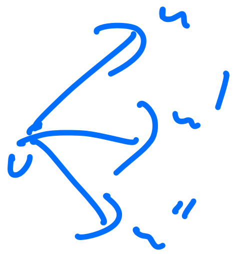
Topological Sort Analysis

Correctness:

Topological Sort Analysis

Correctness: Since G a DAG, never see back edge

Topological Sort Analysis

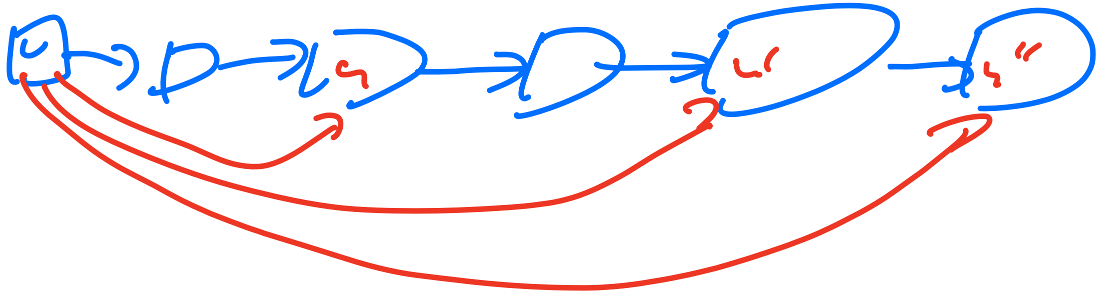


Correctness: Since G a DAG, never see back edge

$\Longrightarrow$ Every edge (v, u) out of v a forward or cross edge

$\Longrightarrow$ $\textit{finish}(u) < \textit{finish}(v)$

$\Longrightarrow$ u already in list when v added to beginning



Topological Sort Analysis

Correctness: Since G a DAG, never see back edge

$\implies$ Every edge (v, u) out of v a forward or cross edge

$\implies$ $\mathit{finish}(u) < \mathit{finish}(v)$

$\implies$ u already in list when v added to beginning

Running Time:

Topological Sort Analysis

Correctness: Since G a DAG, never see back edge

$\Longrightarrow$ Every edge (v, u) out of v a forward or cross edge

$\Longrightarrow$ $\mathit{finish}(u) < \mathit{finish}(v)$

$\Longrightarrow$ u already in list when v added to beginning

Running Time: Same as DFS! $O(m + n)$

Strongly Connected Components (SCC)

Definitions

Another application of DFS. "Kosaraju's Algorithm": Developed by Rao Kosaraju, professor emeritus at JHU CS!

$G = (V, E)$ a directed graph.

Definition

$C \subseteq V$ is a *strongly connected component (SCC)* if it is a *maximal* subset such that for all $u, v \in C$, u can reach v and vice versa (bireachable).

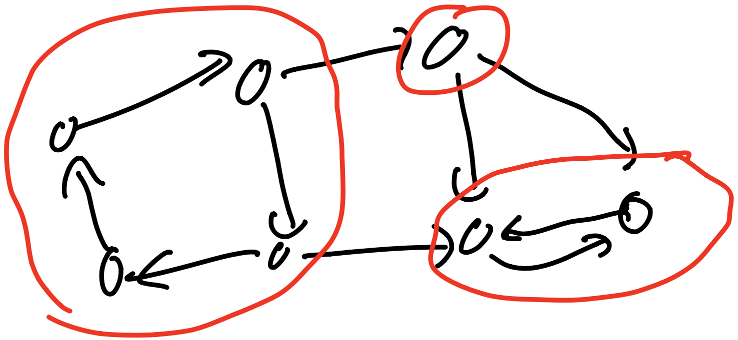
Definitions

Another application of DFS. "Kosaraju's Algorithm": Developed by Rao Kosaraju, professor emeritus at JHU CS!

$\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$ a directed graph.

Definition

$\boldsymbol{C} \subseteq \boldsymbol{V}$ is a *strongly connected component (SCC)* if it is a *maximal* subset such that for all $\boldsymbol{u}, \boldsymbol{v} \in \boldsymbol{C}$, $\boldsymbol{u}$ can reach $\boldsymbol{v}$ and vice versa (bireachable).



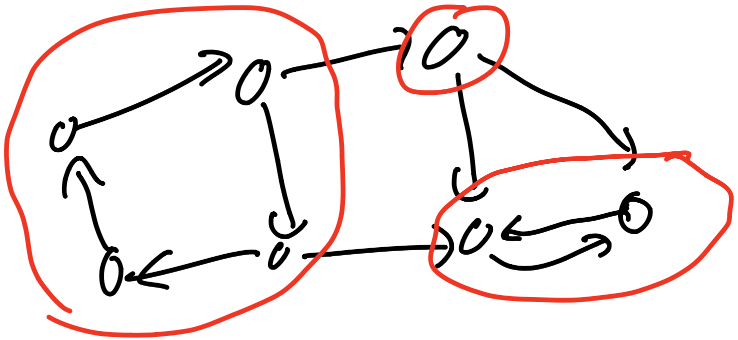
Definitions

Another application of DFS. "Kosaraju's Algorithm": Developed by Rao Kosaraju, professor emeritus at JHU CS!

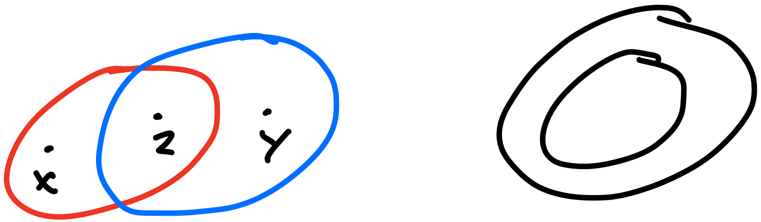
$G = (V, E)$ a directed graph.

Definition

$C \subseteq V$ is a *strongly connected component (SCC)* if it is a *maximal* subset such that for all $u, v \in C$, u can reach v and vice versa (bireachable).



Fact: There is a *unique* partition of V into SCCs



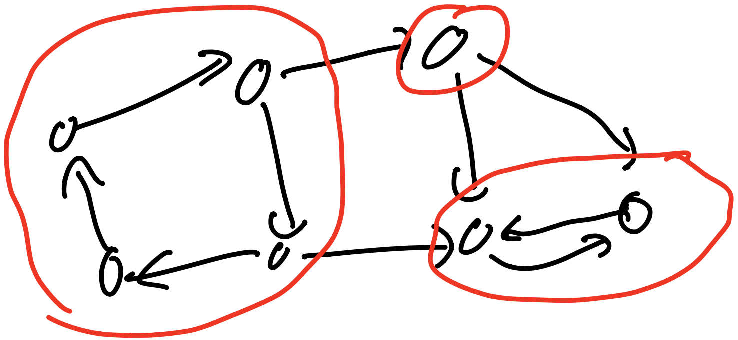
Definitions

Another application of DFS. "Kosaraju's Algorithm": Developed by Rao Kosaraju, professor emeritus at JHU CS!

$G = (V, E)$ a directed graph.

Definition

$C \subseteq V$ is a *strongly connected component (SCC)* if it is a *maximal* subset such that for all $u, v \in C$, u can reach v and vice versa (bireachable).



Fact: There is a *unique* partition of V into SCCs

Proof: Bireachability is an equivalence relation: if u and v are bireachable, and v and w are bireachable, then u and w are bireachable.

SCC Problem

Problem: Given $\boldsymbol{G}$, compute SCCs (partition $\boldsymbol{V}$ into the SCCs)

SCC Problem

Problem: Given G, compute SCCs (partition V into the SCCs)

Trivial Algorithm:

SCC Problem

Problem: Given G, compute SCCs (partition V into the SCCs)

Trivial Algorithm: DFS/BFS from every node, keep track of what's reachable from where

SCC Problem

Problem: Given G, compute SCCs (partition V into the SCCs)

Trivial Algorithm: DFS/BFS from every node, keep track of what's reachable from where

- Running time: $O(n(m+n))$

SCC Problem

Problem: Given G, compute SCCs (partition V into the SCCs)

Trivial Algorithm: DFS/BFS from every node, keep track of what's reachable from where

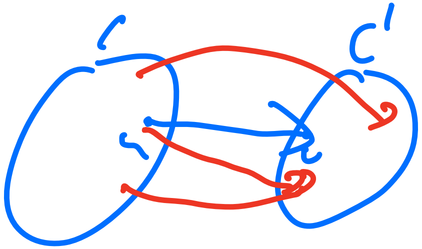
- Running time: $O(n(m+n))$

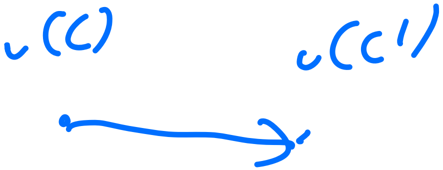
Can we do better? $O(m+n)$?

Graph of SCCs

Definition: Let $\hat{G}$ be graph of SCCs:

- Vertex $v(C)$ for each SCC C
- Edge $(v(C), v(C'))$ if $\exists\ u \in C, v \in C'$ such that $(u, v) \in E$

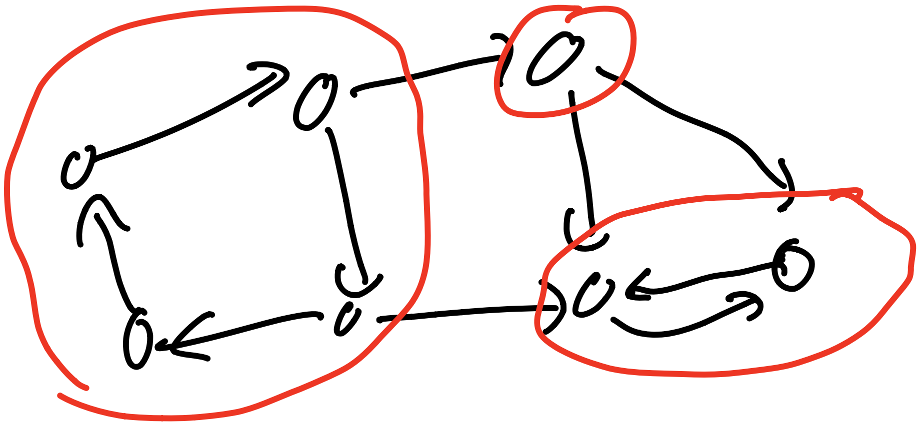


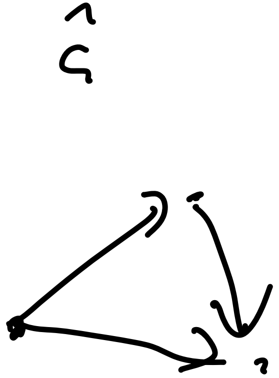


Graph of SCCs

Definition: Let $\hat{G}$ be graph of SCCs:

- Vertex $v(C)$ for each SCC C
- Edge $(v(C), v(C'))$ if $\exists\, u \in C, v \in C'$ such that $(u, v) \in E$





Graph of SCCs: Structure

Theorem

$\hat{G}$ is a DAG.

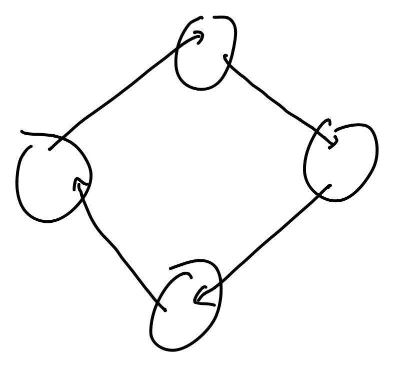
Graph of SCCs: Structure

Theorem

$\hat{G}$ is a DAG.

Proof.

Suppose $\hat{G}$ not a DAG. Then there is a directed cycle H.



□

Graph of SCCs: Structure

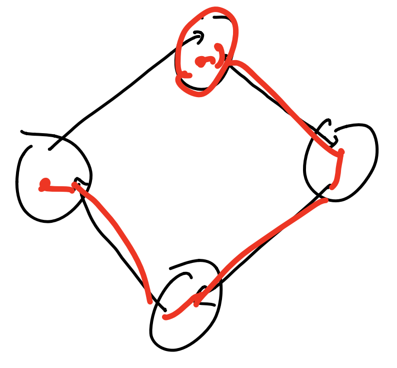
Theorem

$\hat{G}$ *is a DAG.*

Proof.

Suppose $\hat{G}$ not a DAG. Then there is a directed cycle H.

$\implies \bigcup_{C : v(C) \in H} C$ is an SCC



□

Graph of SCCs: Structure

Theorem

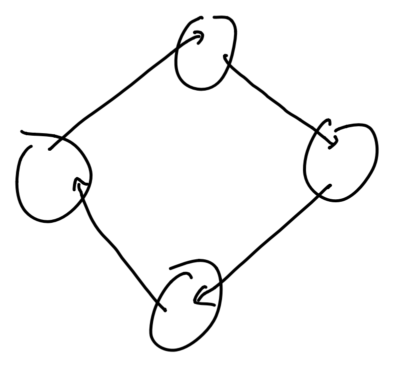
$\hat{G}$ is a DAG.

Proof.

Suppose $\hat{G}$ not a DAG. Then there is a directed cycle H.

$\implies \bigcup_{C:v(C)\in H} C$ is an SCC

$\implies v(C)$ not an SCC for $v(C) \in H$



□

Graph of SCCs: Structure

Theorem

$\hat{G}$ is a DAG.

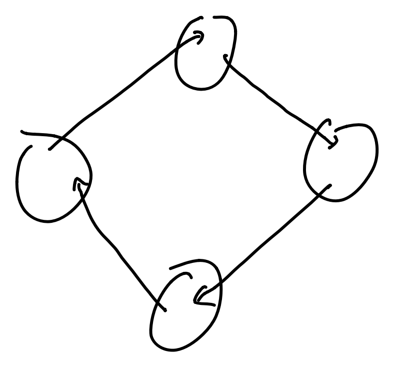
Proof.

Suppose $\hat{G}$ not a DAG. Then there is a directed cycle H.

$\implies \bigcup_{C:v(C)\in H} C$ is an SCC

$\implies v(C)$ not an SCC for $v(C) \in H$

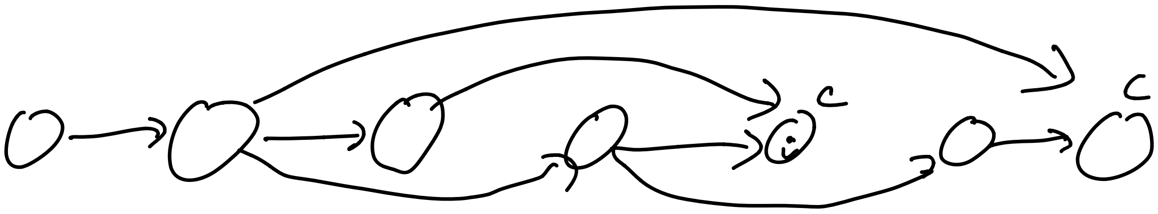
Contradiction!



□

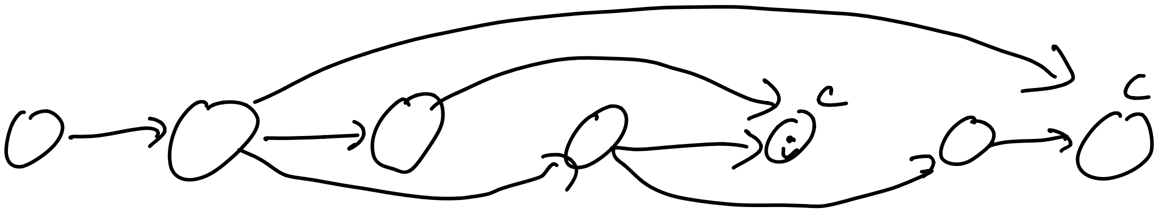
Sink SCC

Since $\hat{G}$ a DAG, has a topological sort



Sink SCC

Since $\hat{G}$ a DAG, has a topological sort

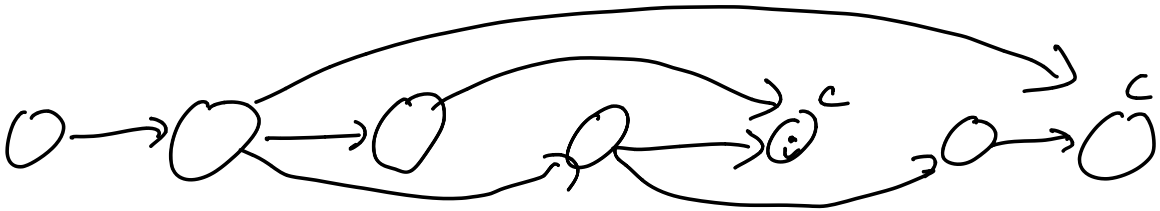


Definition: SCC C is a *sink* SCC if no outgoing edges in $\hat{G}$

- Claim: At least one sink SCC exists

Sink SCC

Since $\hat{G}$ a DAG, has a topological sort

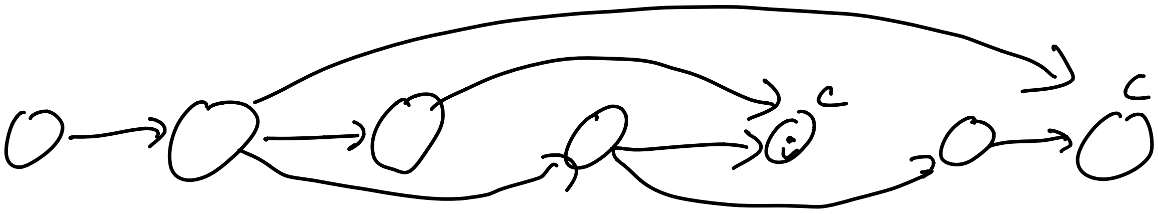


Definition: SCC C is a *sink* SCC if no outgoing edges

- Claim: At least one sink SCC exists
- Proof: Final SCC in topological sort of $\hat{G}$ must be a sink.

Sink SCC

Since $\hat{G}$ a DAG, has a topological sort



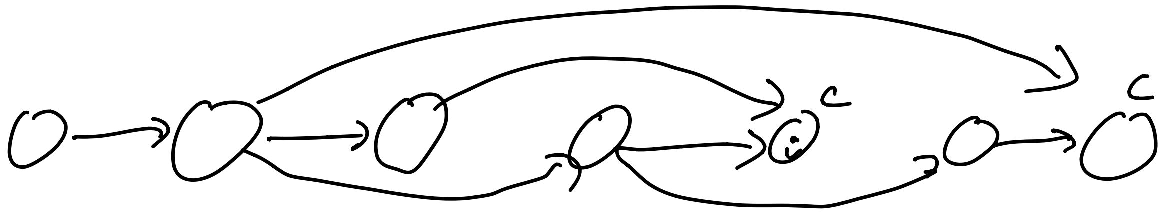
Definition: SCC C is a *sink* SCC if no outgoing edges

- Claim: At least one sink SCC exists
- Proof: Final SCC in topological sort of $\hat{G}$ must be a sink.

What happens if we run DFS(v) where v in a sink SCC?

Sink SCC

Since $\hat{G}$ a DAG, has a topological sort



Definition: SCC C is a *sink* SCC if no outgoing edges

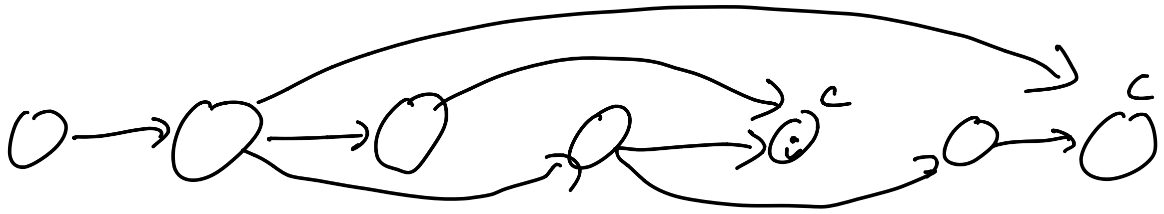
- Claim: At least one sink SCC exists
- Proof: Final SCC in topological sort of $\hat{G}$ must be a sink.

What happens if we run DFS(v) where v in a sink SCC?

- See exactly nodes in C!

Sink SCC

Since $\hat{G}$ a DAG, has a topological sort



Definition: SCC C is a *sink* SCC if no outgoing edges

- Claim: At least one sink SCC exists
- Proof: Final SCC in topological sort of $\hat{G}$ must be a sink.

What happens if we run DFS(v) where v in a sink SCC?

- See exactly nodes in C!

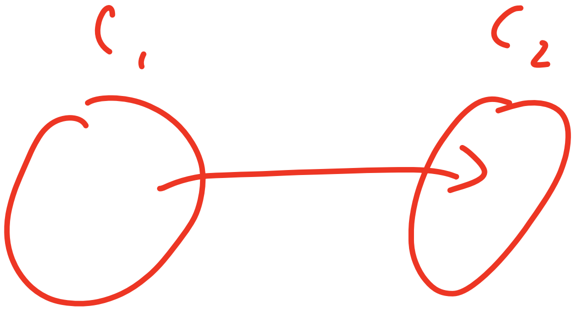
Strategy: find node in sink SCC, run DFS, remove nodes found, repeat

SCCs and DFS

Run DFS(G), and let $\textit{finish}(C) = \max_{v \in C} \textit{finish}(v)$

Lemma

Let C_1, C_2 distinct SCCs s.t. $(v(C_1), v(C_2)) \in E(\hat{G})$. Then $\textit{finish}(C_1) > \textit{finish}(C_2)$.



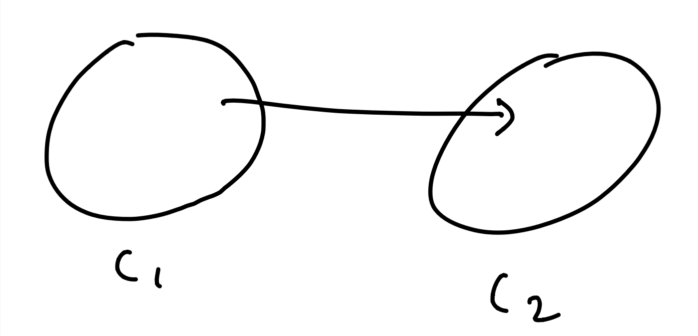
SCCs and DFS

Run DFS(G), and let $\textit{finish}(C) = \max_{v \in C} \textit{finish}(v)$

Lemma

Let C_1, C_2 distinct SCCs s.t. $(v(C_1), v(C_2)) \in E(\hat{G})$. Then $\textit{finish}(C_1) > \textit{finish}(C_2)$.

Proof.



Let $x \in C_1 \cup C_2$ be first node encountered by DFS

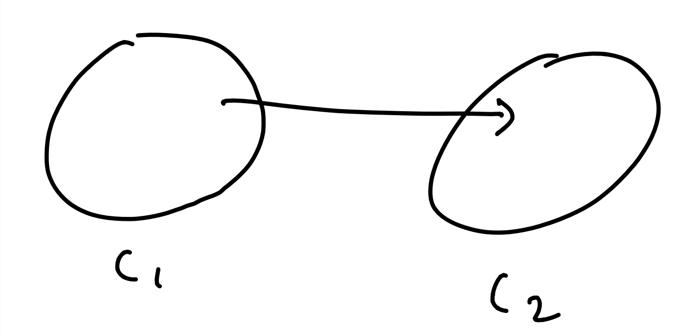
SCCs and DFS

Run DFS(G), and let $\textbf{finish}(C) = \max_{v \in C} \textbf{finish}(v)$

Lemma

Let C_1, C_2 distinct SCCs s.t. $(v(C_1), v(C_2)) \in E(\hat{G})$. Then $\textbf{finish}(C_1) > \textbf{finish}(C_2)$.

Proof.



Let $x \in C_1 \cup C_2$ be first node encountered by DFS

- If $x \in C_1$:

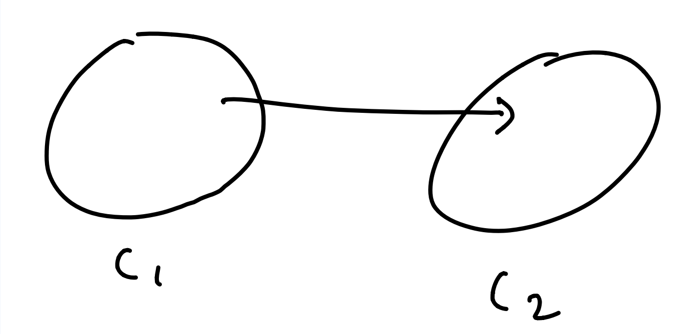
SCCs and DFS

Run DFS(G), and let $\textit{finish}(C) = \max_{v \in C} \textit{finish}(v)$

Lemma

Let C_1, C_2 distinct SCCs s.t. $(v(C_1), v(C_2)) \in E(\hat{G})$. Then $\textit{finish}(C_1) > \textit{finish}(C_2)$.

Proof.



Let $x \in C_1 \cup C_2$ be first node encountered by DFS

- If $x \in C_1$: all of C_2 reachable from x, so DFS(x) does not complete until all of C_2 finished

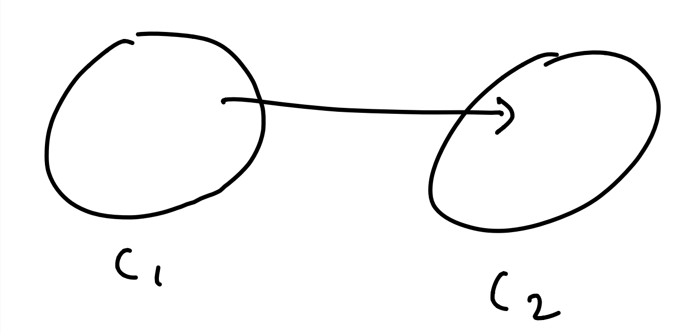
SCCs and DFS

Run DFS(G), and let $\textit{finish}(C) = \max_{v \in C} \textit{finish}(v)$

Lemma

Let C_1, C_2 distinct SCCs s.t. $(v(C_1), v(C_2)) \in E(\hat{G})$. Then $\textit{finish}(C_1) > \textit{finish}(C_2)$.

Proof.



Let $x \in C_1 \cup C_2$ be first node encountered by DFS

- If $x \in C_1$: all of C_2 reachable from x, so DFS(x) does not complete until all of C_2 finished
- If $x \in C_2$:

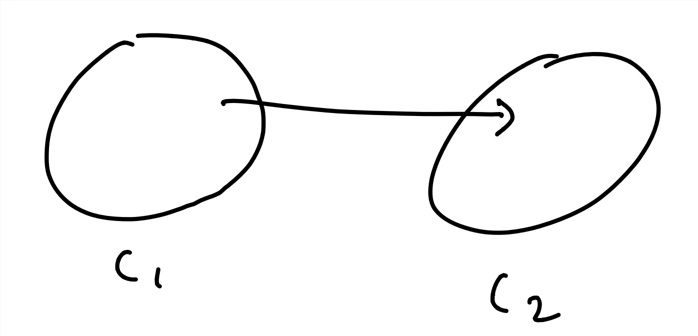
SCCs and DFS

Run DFS(G), and let $\textit{finish}(C) = \max_{v \in C} \textit{finish}(v)$

Lemma

Let C_1, C_2 distinct SCCs s.t. $(v(C_1), v(C_2)) \in E(\hat{G})$. Then $\textit{finish}(C_1) > \textit{finish}(C_2)$.

Proof.

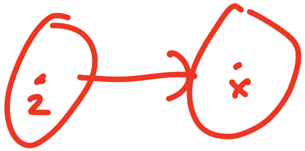


Let $x \in C_1 \cup C_2$ be first node encountered by DFS

- If $x \in C_1$: all of C_2 reachable from x, so DFS(x) does not complete until all of C_2 finished
- If $x \in C_2$: all of C_2 reachable from x but nothing from C_1, so all of C_2 finishes before any node in C_1 starts

□

SCCs and DFS

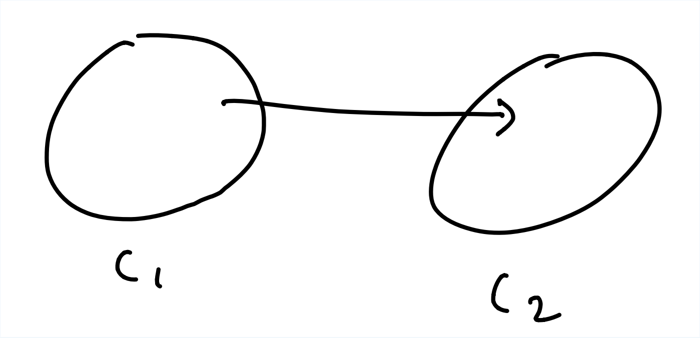


Run DFS(G), and let $\textit{finish}(C) = \max_{v \in C} \textit{finish}(v)$

Lemma

Let C_1, C_2 distinct SCCs s.t. $(v(C_1), v(C_2)) \in E(\hat{G})$. Then $\textit{finish}(C_1) > \textit{finish}(C_2)$.

Proof.



Let $x \in C_1 \cup C_2$ be first node encountered by DFS

- If $x \in C_1$: all of C_2 reachable from x, so DFS(x) does not complete until all of C_2 finished
- If $x \in C_2$: all of C_2 reachable from x but nothing from C_1, so all of C_2 finishes before any node in C_1 starts

□

So node of *max* finishing time in a *source* SCC (no incoming edges in $\hat{G}$).

Useful Corollary

Run DFS(G), and let $\textit{finish}(C) = \max_{v \in C} \textit{finish}(v)$.

Corollary

Let $\mathcal{C}$ be collection of all SCCs of G, and let $\mathcal{C}' \subseteq \mathcal{C}$. Let $G' = G \setminus (\bigcup_{C \in \mathcal{C}'} C)$. Then the node $v = \text{argmax}_{u \in \bigcup_{C \in \mathcal{C} \setminus \mathcal{C}'} C} \textit{finish}(u)$ is in an SCC of G that is a source SCC of G'.

Useful Corollary

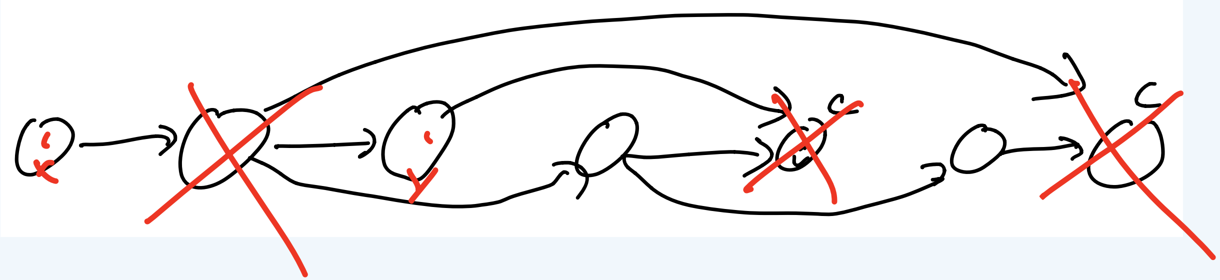
Run DFS(G), and let $\textit{finish}(C) = \max_{v \in C} \textit{finish}(v)$.

Corollary

Let $\mathcal{C}$ be collection of all SCCs of G, and let $\mathcal{C}' \subseteq \mathcal{C}$. Let $G' = G \setminus (\bigcup_{C \in \mathcal{C}'} C)$. Then the node $v = \mathrm{argmax}_{u \in \bigcup_{C \in \mathcal{C} \setminus \mathcal{C}'} C} \textit{finish}(u)$ is in an SCC of G that is a source SCC of G'.

Proof.

Clearly SCCs of G' are precisely $\mathcal{C} \setminus \mathcal{C}'$:



Useful Corollary

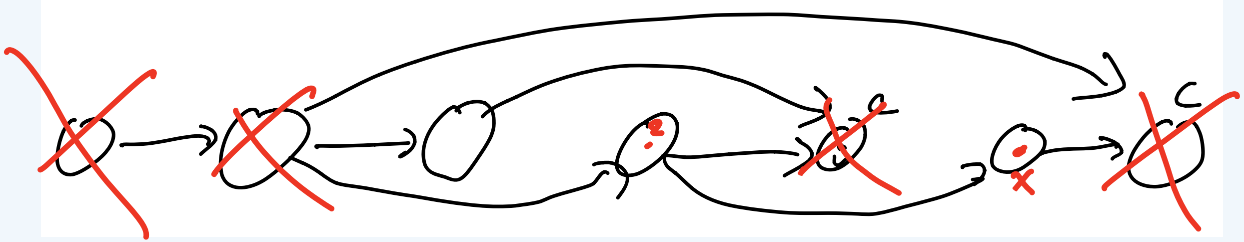
Run DFS(G), and let $\textit{finish}(C) = \max_{v \in C} \textit{finish}(v)$.

Corollary

Let $\mathcal{C}$ be collection of all SCCs of G, and let $\mathcal{C}' \subseteq \mathcal{C}$. Let $G' = G \setminus (\bigcup_{C \in \mathcal{C}'} C)$. Then the node $v = \text{argmax}_{u \in \bigcup_{C \in \mathcal{C} \setminus \mathcal{C}'} C} \textit{finish}(u)$ is in an SCC of G that is a source SCC of G'.

Proof.

Clearly SCCs of G' are precisely $\mathcal{C} \setminus \mathcal{C}'$:



Lemma $\implies$ node remaining with max finish time in a ~~sink~~ source SCC of what remains. □

Kosaraju's Algorithm

Kosaraju's Algorithm

So node with max finish time in a *source* SCC (no incoming edges in $\hat{\boldsymbol{G}}$). Want sink (no outgoing edges).

Kosaraju's Algorithm

So node with max finish time in a *source* SCC (no incoming edges in $\hat{\boldsymbol{G}}$). Want sink (no outgoing edges). Reverse all edges!

Kosaraju's Algorithm

So node with max finish time in a *source* SCC (no incoming edges in $\hat{G}$). Want sink (no outgoing edges). Reverse all edges!

Definition: G^T is G with all edges reversed.

- Source SCC in G^T is sink SCC in G

Kosaraju's Algorithm

So node with max finish time in a *source* SCC (no incoming edges in $\hat{G}$). Want sink (no outgoing edges). Reverse all edges!

Definition: G^T is G with all edges reversed.

- Source SCC in G^T is sink SCC in G

Kosaraju's Algorithm:

- DFS(G^T) to get finishing times and order π on V from smallest finishing time to largest
- Set $mark(v) = False$ for all $v \in V$
- Forall $v \in V$ in order of π {

    if $mark(v) = False$ {

        Run DFS(v), let C be all nodes found

        Return C as an SCC

    }

}

Kosaraju's Algorithm

So node with max finish time in a *source* SCC (no incoming edges in $\hat{G}$). Want sink (no outgoing edges). Reverse all edges!

Definition: G^T is G with all edges reversed.

- Source SCC in G^T is sink SCC in G

Kosaraju's Algorithm:

- DFS(G^T) to get finishing times and order π on V from smallest finishing time to largest
- Set $mark(v) = False$ for all $v \in V$
- Forall $v \in V$ in order of π {
    if $mark(v) = False$ {
        Run DFS(v), let C be all nodes found
        Return C as an SCC
    }
  }

Running Time:

Kosaraju's Algorithm

So node with max finish time in a *source* SCC (no incoming edges in $\hat{G}$). Want sink (no outgoing edges). Reverse all edges!

Definition: G^T is G with all edges reversed.

- Source SCC in G^T is sink SCC in G

Kosaraju's Algorithm:

- DFS(G^T) to get finishing times and order π on V from smallest finishing time to largest
- Set $mark(v) = False$ for all $v \in V$
- Forall $v \in V$ in order of π {

  if $mark(v) = False$ {

  Run DFS(v), let C be all nodes found

  Return C as an SCC

  }

}

Running Time: $O(m + n)$

Correctness Sketch

Let $C_1, C_2, \ldots, C_k$ be sets identified by algorithm (in order)

Theorem

C_i is a sink SCC of $G \setminus \left(\bigcup_{j=1}^{i-1} C_j\right)$, and an SCC of G.

Correctness Sketch

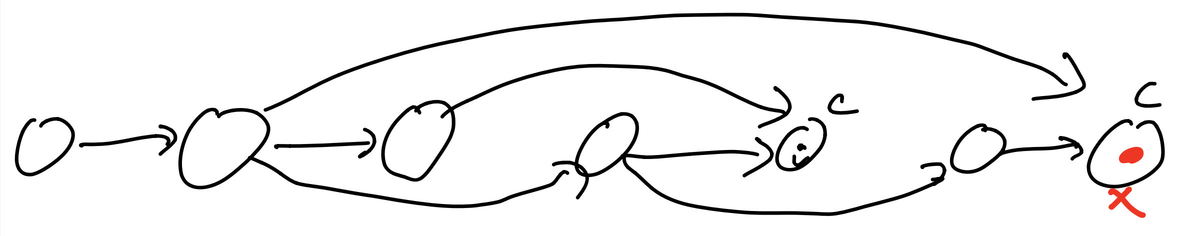
Let $C_1, C_2, \ldots, C_k$ be sets identified by algorithm (in order)

Theorem

C_i is a sink SCC of $G \setminus \left(\bigcup_{j=1}^{i-1} C_j\right)$, and an SCC of G.

Proof Sketch.

Induction on i.



Correctness Sketch

Let $C_1, C_2, \ldots, C_k$ be sets identified by algorithm (in order)

Theorem

C_i is a sink SCC of $G \setminus \left(\bigcup_{j=1}^{i-1} C_j\right)$, and an SCC of G.

Proof Sketch.

Induction on i.

Base case: $i = 1$. By previous argument, largest finishing time in $G^T \implies$ in sink SCC of G $\implies C_1$ is sink SCC of G

Correctness Sketch

Let $C_1, C_2, \ldots, C_k$ be sets identified by algorithm (in order)

Theorem

C_i is a sink SCC of $G \setminus \left(\bigcup_{j=1}^{i-1} C_j\right)$, and an SCC of G.

Proof Sketch.

Induction on i.

Base case: $i = 1$. By previous argument, largest finishing time in $G^T \implies$ in sink SCC of G $\implies C_1$ is sink SCC of G

Inductive case: Let $i > 1$. Let v unmarked node with largest finishing time.

- By induction, subgraph of unmarked nodes is G minus $i - 1$ SCCs of G
- Corollary $\implies v$ must be in sink SCC of unmarked nodes so get an SCC of unmarked nodes when run DFS
- Corollary $\implies$ SCC of original graph