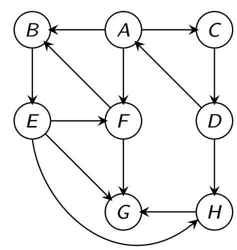
Chapter 2

DFS in Directed Graphs, Strong Connected Components, and DAGs

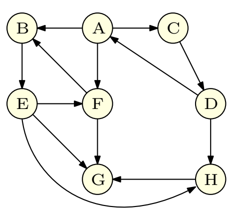
OLD CS 473: Fundamental Algorithms, Spring 2015
January 22, 2015

2.0.0.1 Strong Connected Components (SCCs)

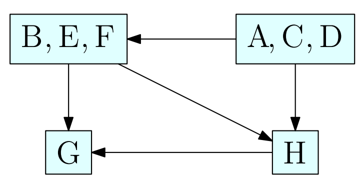
Algorithmic Problem Find all SCCs of a given directed graph. Previous lecture:
Saw an $O(n \cdot (n+m))$ time algorithm.
This lecture: $O(n+m)$ time algorithm.



2.0.0.2 Graph of SCCs



Graph G



$\mathsf{G}^{\mathrm{SCC}}$: Graph of SCCs

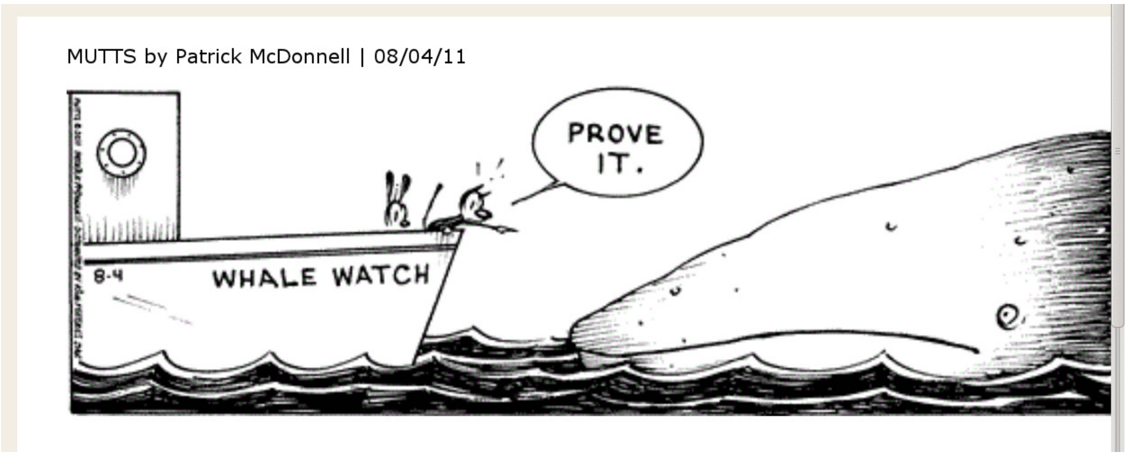
Meta-graph of SCCs Let $S_1, S_2, \ldots S_k$ be the strong connected components (i.e., SCCs) of G. The graph of SCCs is $\mathsf{G}^{\mathrm{SCC}}$

(A) Vertices are $S_1, S_2, \ldots S_k$
(B) There is an edge (S_i, S_j) if there is some $u \in S_i$ and $v \in S_j$ such that (u, v) is an edge in G.

2.0.0.3 Reversal and SCCs

Proposition 2.0.1. *For any graph G, the graph of SCCs of G^{rev} is the same as the reversal of G^{SCC}.*

Proof: Exercise. ■



2.0.0.4 SCCs and DAGs

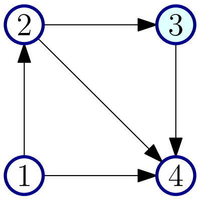
Proposition 2.0.2. *For any graph* G*, the graph* $\mathsf{G}^{\mathrm{SCC}}$ *has no directed cycle.*

Proof: If $\mathsf{G}^{\mathrm{SCC}}$ has a cycle $S_1, S_2, \ldots, S_k$ then $S_1 \cup S_2 \cup \cdots \cup S_k$ should be in the same SCC in G. Formal details: exercise. ■

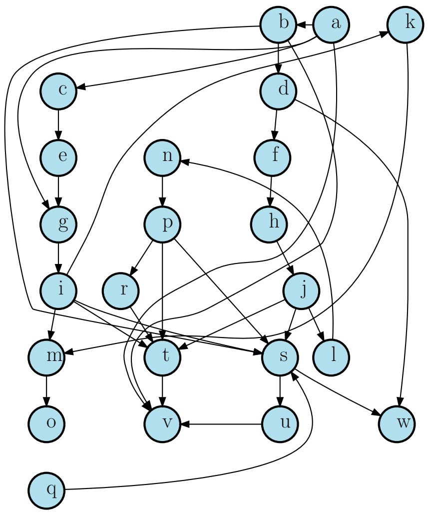
2.1 Directed Acyclic Graphs

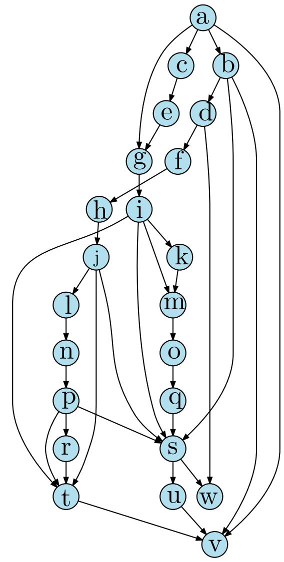
2.1.0.5 Directed Acyclic Graphs

Definition 2.1.1. *A directed graph* G *is a* **directed acyclic graph** *(DAG) if there is no directed cycle in* G*.*

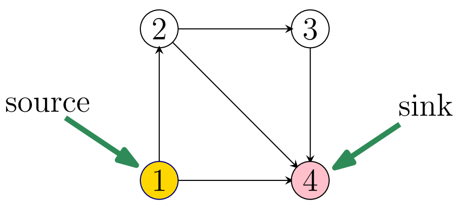


2.1.0.6 Is this a DAG?





2.1.0.7 Sources and Sinks



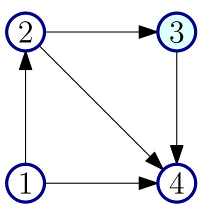
Definition 2.1.2. *(A) A vertex u is a* **source** *if it has no in-coming edges.*
(B) A vertex u is a **sink** *if it has no out-going edges.*

2.1.0.8 Simple DAG Properties

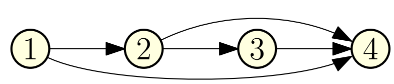
(A) Every DAG G has at least one source and at least one sink.
(B) If G is a DAG if and only if $\mathsf{G}^{\mathrm{rev}}$ is a DAG.
(C) G is a DAG if and only each node is in its own strong connected component.

Formal proofs: exercise.

2.1.0.9 Topological Ordering/Sorting



Graph G



Topological Ordering of G

Definition 2.1.3. *A* **topological ordering/topological sorting** *of $G = (V, E)$ is an ordering $\prec$ on V such that if $(u, v) \in E$ then $u \prec v$.*

Informal equivalent definition: One can order the vertices of the graph along a line (say the x-axis) such that all edges are from left to right.

2.1.0.10 DAGs and Topological Sort

Lemma 2.1.4. *A directed graph* G *can be topologically ordered iff it is a* DAG*.*

Proof: $\Longrightarrow$: Suppose G is not a DAG and has a topological ordering $\prec$. G has a cycle $C = u_1, u_2, \ldots, u_k, u_1$.

Then $u_1 \prec u_2 \prec \ldots \prec u_k \prec u_1$!

That is... $u_1 \prec u_1$.

A contradiction (to $\prec$ being an order).

Not possible to topologically order the vertices. ∎

2.1.0.11 DAGs and Topological Sort

Lemma 2.1.5. *A directed graph* G *can be topologically ordered iff it is a* DAG*.*

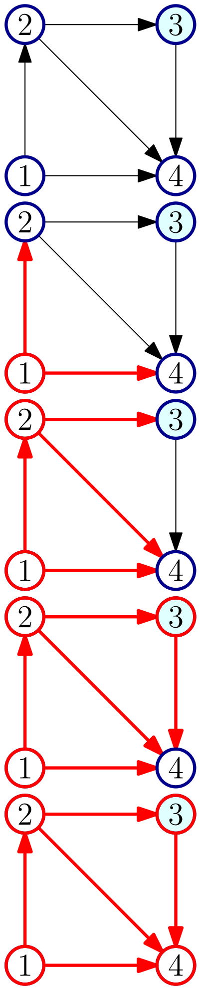
Proof:[Continued] $\Leftarrow$: Consider the following algorithm:

(A) Pick a source u, output it.
(B) Remove u and all edges out of u.
(C) Repeat until graph is empty.
(D) Exercise: prove this gives an ordering.

∎

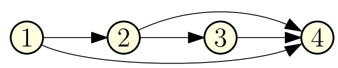
Exercise: show above algorithm can be implemented in $O(m + n)$ time.

2.1.0.12 Topological Sort: An Example



Output: 1 2 3 4

2.1.0.13 Topological Sort: Another Example



2.1.0.14 DAGs and Topological Sort

Note: A DAG G may have many different topological sorts.

Question: What is a DAG with the most number of distinct topological sorts for a given number n of vertices?

Question: What is a DAG with the least number of distinct topological sorts for a given number n of vertices?