Directed graphs (digraphs)

Edge \((u, v)\) is directed from \(u\) to \(v\).

\(u\) and \(v\) are **reachable** if there is a directed path from \(u\) to \(v\). That is, a sequence of nodes \((u = v_0, v_1, v_2, \ldots, v_k = v)\), such that \((v_i, v_{i+1})\) are directed edges.

In notation  \(u \rightarrow v\)

Reachability is transitive.  \((u \rightarrow v) \land (v \rightarrow w) = (u \rightarrow w)\)

**concatenate paths**  \(u \rightarrow v\) with \(v \rightarrow w\) to get  \(u \rightarrow w\)
Directed acyclic graphs (DAG)

A directed acyclic graph (DAG) is a directed graph that contains no directed cycles.

- a directed cycle is a sequence of vertices $v_0, v_1, .., v_k$ so that there are directed edges between subsequent nodes $(v_i, v_{i+1})$ and an edge $(v_k, v_0)$.

Exercise. Show that a graph $G$ is a DAG iff ("if and only if") $(u \rightarrow v) \land (v \rightarrow u)$ implies $u = v$.

Suppose $u$ is different from $v$, then the two paths together ($u \rightarrow v$ and $v \rightarrow u$) would form a cycle.
Directed acyclic graphs (DAG)

A directed acyclic graph (DAG) is a directed graph that contains no directed cycles.

“Typical use” of DAGs: directed edges mean some kind of precedence constraint.
- course prerequisites: CS111, CS112 and CS131 must be taken before CS330
- compilation: module $v_i$ must be compiled before module $v_j$
- order of cloths when getting dressed.
  - note that there may be vertices that are independent (neither is a precedence to the other)

Question. Given the tasks and their precedence. How do we know in which order to execute these tasks?
Topological order in DAGs

A **topological order** of a DAG $G(V,E)$ is an ordering of its vertices $v_1^t < v_2^t < ... < v_n^t$ so that for every edge $(v_i,v_j)$ we have $i < j$. I.e., each edge is directed from an earlier to a later node in this order.
Topological order in DAGs - TopHat

A topological order of a DAG $G(V,E)$ is an ordering of its vertices $v_1^f < v_2^f < \ldots < v_n^f$ so that for every edge $(v_i,v_j)$ we have $i < j$. I.e., each edge is directed from an earlier to a later node in this order.

Question. (multiple choice) Here are some possible orders of vertices of the graph on the left. Select all that are valid topological orders.

A. $v_1, v_3, v_2, v_5, v_6, v_7, v_4$
B. $v_1, v_2, v_3, v_4, v_5, v_6, v_7$  ✔
C. $v_3, v_4, v_5, v_6, v_7, v_1, v_2$
D. $v_2, v_1, v_3, v_4, v_5, v_6, v_7$  ✔
Topological order in DAGs

A topological order of a DAG $G(V,E)$ is an ordering of its vertices $v_1^f < v_2^f < \ldots < v_n^f$ so that for every edge $(v_i,v_j)$ we have $i < j$. I.e., each edge is directed from an earlier to a later node in this order.

In this example $v_2, v_1, v_3, \ldots, v_7$ and $v_1, v_2, v_3, \ldots, v_7$ are both valid topological orders.

How to find a topological order?
1. Find a topological order of this graph (by looking at it)!

2. Run DFS on this graph and record start and finish times!

**Question.** Which of these statements is true?

A. vertices ordered by *earliest discovery* time first form a top. order
B. vertices ordered by *latest discovery* time first form a top. order
C. vertices ordered by *earliest finish* time first form a top. order
D. vertices ordered by *latest finish* time first form a topological order
Find the connected components in an undirected graph G.

Algorithm: repeatedly run BFS/DFS from a yet undiscovered node to find the next connected component.
Strongly connected component (SCC)

- In a directed $G$ nodes $u$ and $v$ are strongly connected if $(u \to v)$ and $(v \to u)$
- In a strongly connected component of $G$ all nodes are mutually strongly connected.
- $G$ is strongly connected if it consists of only one SCC.
Strongly connected component (SCC)

5 strongly connected components (3 of those are singletons)

Any node in the SCC can be reached from any other node in the SCC.
Select all the true statements about strongly connected components (SCCs) in a graph

A. Some SCCs consist of a single node ✓
B. non-singleton (consist of more than 1 node) SCCs contain cycles ✓
C. If nodes u and v are in different SCCs, then v is never reachable from u ✗
D. It’s possible for two different SCCs to share a node ✗
E. If we contract the nodes of each SCC into a separate single node (see picture), then the resulting graph is a DAG ✓
Observation. Two SCCs are either disjoint or equal.
Observation. Two SCCs are either disjoint or equal.

If we contract the SC components in one node we get an acyclic graph.
DAGs and topological order

Order the vertices by latest finish time first:

e - g - c - b - d - a - h - f

Claim: latest finish time first algorithm always finds a topological order.

If G is a D.A.G.
Suppose that we run DFS on directed graph starting from a node with 0 indegree. We know that the graph contains a directed cycle as soon as we find an edge \((u,v)\) such that …

A. \(v\) has been discovered but not finished yet.
B. \(v\) has a later discovery time than \(u\).
C. \(v\) has already been finished.

because \(v\) is not finished yet, we know that \(u\) belongs to the subtree under \(v\)

The path from \(v\) to \(u\) plus the edge \(u\rightarrow v\) form a cycle.
Detect cycles in a graph

Suppose $G$ is not a DAG, nevertheless we feed it to the latest finish time first algorithm. How can we use it to detect that $G$ is not a DAG?

We can run the D.F.S. algorithm anyway. If we find a neighbor that was discovered but it is not finished, just stop because it means we have a cycle.
Topological order with DFS

Let $G(V,E)$ be a DAG

Algorithm: sorting vertices by reversed order of DFS finish time, i.e. latest finish time first, yields a topological order

Exercise: Formulate what it means that this algorithm is correct. (make a statement about the edge directions)

It means that for every edge $(u,v)$, $u$ comes before $v$ in the topological order.

In terms of finish times, it means

$u.f > v.f$
Topological order with DFS

Let G(V,E) be a DAG

Algorithm: sorting vertices by reversed order of DFS finish time, i.e. latest finish time first, yields a topological order

Exercise: Formulate what it means that this algorithm is correct.

Lemma. If G is a DAG and (u,v) is a directed edge then $u.f > v.f$.

pf. By contradiction, suppose $v$ is finished later, then the edge $(u,v)$ would be in the opposite direction, as required by the top. order.
DAGs and topological order

Theorem: A graph $G(V,E)$ has a topological order if and only if it is a DAG.

pf.: Proving in two steps

topological order $\Rightarrow$ DAG: next slide

DAG $\Rightarrow$ topological order:
we develop an(other) algorithm to find the topological order. This algorithm will yield the proof.
Lemma. If $G$ has a topological order, then $G$ is a DAG.

Pf. [by contradiction]

Suppose $G$ contains a cycle $C$. Assume that $v_i$ is the first in the order among the nodes in $C$. Let $(v_j,v_i)$ be the last edge on the cycle $C$. Since $v_i$ is the first node of $C$ in the order, this edge is point backwards, therefore, $G$ does not have a top. order. Contradiction!

the supposed topological order: $v_1, \ldots, v_n$
DAGs and the topological order

Lemma. If $G$ has a topological order, then $G$ is a DAG.

Pf. [by contradiction]

- Suppose that $G$ has a topological order $v_1, v_2, \ldots, v_n$ and that $G$ also has a directed cycle $C$. Let’s see what happens.
- Let $v_i$ be the lowest-indexed node in $C$, and let $v_j$ be the node just before $v_i$ in $C$; thus $(v_j, v_i)$ is an edge.
- By our choice of $i$, we have $i < j$.
- On the other hand, since $(v_j, v_i)$ is an edge and $v_1, v_2, \ldots, v_n$ is a topological order, we must have $j < i$, a contradiction. □
Topological order by in-degree

Lemma. Let $G(V,E)$ be a DAG. Then there is always a node $v$ with $\text{in-degree}(v) = 0$.

Pf. [by contradiction]

- Suppose every node has indegree $\geq 1$:
  1. Pick any node at random
  2. Backtrack along the incoming edges
  3. The graph has $n$ nodes, so after at most $n$ steps we visit a node for the second time.
Topological order by in-degree

Lemma. Let \( G(V,E) \) be a DAG. Then there is always a node \( v \) with \( \text{in-degree}(v) = 0 \).

Pf. [by contradiction]

- Suppose that \( G \) is a DAG and every node has at least one entering edge. Let’s see what happens.
- Pick any node \( v \), and begin following edges backward from \( v \). Since \( v \) has at least one entering edge \((u,v)\) we can walk backward to \( u \).
- Then, since \( u \) has at least one entering edge \((x,u)\), we can walk backward to \( x \).
- Repeat until we visit a node, say \( w \), twice.
- Let \( C \) denote the sequence of nodes encountered between successive visits to \( w \). \( C \) is a cycle. ▪
Suppose $G(V,E)$ is a DAG and $\text{in-degree}(v_0) = 0$.

We delete $v_0$ and its adjacent edges from $G$ to get $G'$. Is $G'$ a DAG?

Yes. Deleting edges does not create cycles.

What can we say about the in-degrees of nodes in $G'$?

$G'$ also has a 0-indegree node.

Algorithm:
1. Pick any 0-indegree vertex.
2. Put $v$ in the topological order output.
3. Delete $v$ and its edges.
4. Repeat 1. until there is no vertex left.
Zero in-degree algorithm is correct

**Theorem:** if G is a DAG then the zero in-degree algorithm returns a topological order of the vertices.

pf. [by induction on the number of vertices]

- The theorem is clearly true for |V| = 1.
- Suppose that for any DAG with |V| = n-1 vertices, the algorithm returns a valid topological order.
- Let G(V,E) be a DAG with n vertices.
- Since G is a DAG, by our lemma it has at least one node v with in-degree(v) = 0.
- Let G’ be the graph that we get by deleting v and its adjacent edges from G. Since G’ is a DAG with n-1 nodes, we know by the inductive assumption that the algorithm yields a valid topological order for it. Let this order be v_1,v_2,..,v_{n-1}
- Adding v as the first element to the order v, v_1,v_2,..,v_{n-1} we claim that this is still a topological order.
- In a topological order every edge is directed from a lower to higher indexed node. We only need to show that this is true for edges that v is involved.
  - since v is the first in the order, every edge directed away from v satisfies the rule
  - v has 0 in-degree, so there are no edges directed towards v
Exercise 1: We are given the directed adjacency list of a directed graph $G$.
- We can compute the out-degree of each node as the length of their corresponding neighbor list. How can we compute the in-degree of each vertex in time $O(n+m)$?

Reverse the adjacency list $G$

1. Create an empty adjacency list $G^r$
2. Iterate through each node $v$ of $G$.
3. For every neighbor $w$ of $v$, create a record on $G^r$, using $w$ as the key in the main list and $v$ as the neighbor node in the adjacency list $G^r$