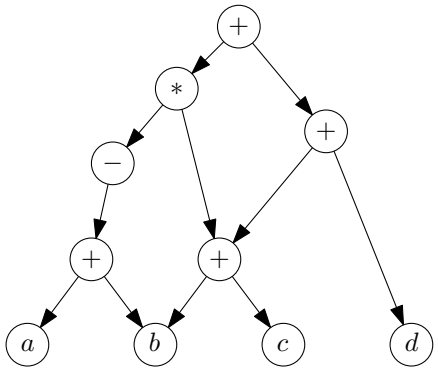
Lecture 26

Topological sort, acyclic graphs and girth

Many computer science applications require us to find precedence (or dependencies) among events. If we consider the events to be nodes and an arc (u, v) means that u precedes v, that is, that u must be calculated before v can be calculated, deciding the order in which to process events becomes a problem of sorting the nodes of a digraph.

Example 26.1. Consider a compiler evaluating sub-expressions of the expression $-(a+b)*(b+c)+((b+c)+d)$. The compiler must compute, for example, $(a + b)$ and $(b + c)$ before it could compute $-(a + b) * (b + c)$. This can be shown as a dependency digraph where the arc (u, v) means that u depends on v so that v must be calculated first (this is the opposite of a precedence digraph discussed above).

$-(a+b)*(b+c)+((b+c)+d)$



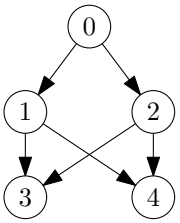
Respecting all precedences/dependencies is equivalent to drawing the digraph with all nodes in a straight line and the arcs all pointing the same direction. The order of calculation starts at one end of the line and proceeds to the other.

26.1 Topological sort

Definition 26.2. Let G be a digraph. A ***topological sort*** of G is a linear ordering of all its vertices such that if G contains an arc (u,v), then u appears before v in the ordering. It is also known as a ***topological order*** or ***linear order***.

For our arithmetic expression example above, a linear order of the sub-expression digraph gives us an order (actually the reverse of the order) where we can safely evaluate the expression.

Example 26.3. A digraph with all possible topological orders and drawn with topological order $0, 1, 2, 3, 4$. Draw the digraph for the topological order $0, 2, 1, 4, 3$.

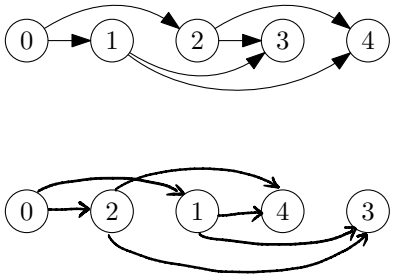


$0, 1, 2, 3, 4$

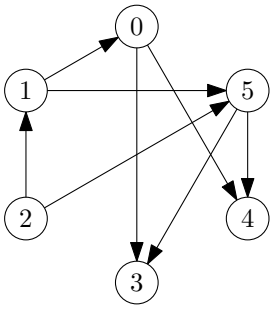
$0, 1, 2, 4, 3$

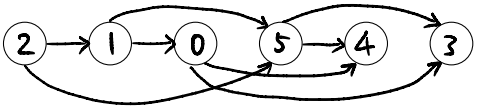
$0, 2, 1, 3, 4$

$0, 2, 1, 4, 3$

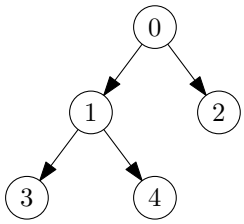


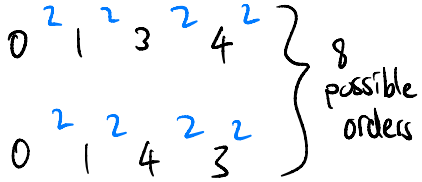
Find a topological order of the following digraph and draw it.





Topological orders are not unique. List all possible topological orders of the following digraph.





If the digraph contains a cycle, it is not possible to find such a linear ordering. This corresponds to inconsistencies in the precedences

given (for example, a precedes b precedes c precedes a) and no scheduling of the tasks is possible.

Definition 26.4. A digraph without cycles is commonly called a ***DAG***, an abbreviation for **d**irected **a**cyclic **g**raph.

Theorem 26.5. A digraph has a topological order if and only if it is a DAG.

Example 26.6. It is clear that if a digraph has a topological order it has no cycles, so it a DAG. Show that a DAG always has a topological order by first showing that every DAG has a source and that by removing the source and any out-arcs from the source you still have a DAG. Show that the order in which nodes are removed gives a topological order.

source is a node with no in-neighbors

- Consider DAG G (recall G is finite i.e. $|V(G)| = n < \infty$)
- Suppose G has no source. G_r has no sink.
- So can take a walk in G that never repeats any nodes and never stops because we never meet a sink.
- So G has a source, eg. u. So u can start our topological order.
- Now consider $G - \{u\}$. It's still a DAG, so must have source

G_r = reverse of G

G_r is also a DAG

This theorem and proof then gives an algorithm ***zero-indegree sorting*** for topologically sorting a DAG. If we apply zero-indegree sorting to a digraph that is not a DAG, eventually it will stop because no source node can be found at some point.

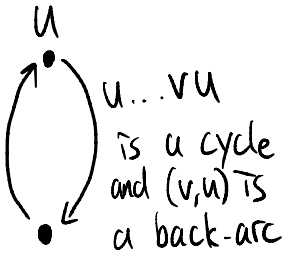
n = visit list
m = search content

Example 26.7. What is the running time of a naive implementation of zero-indegree where a source is found and then removed at each step? How could this idea be made more efficient?

- Finding in-degree is $\Theta(n+m)$, need to perform n times $O(n(n+m))$
- But can find all in-degrees in time $\Theta(n+m)$ → update as we build the order

$\Theta(n+m)$

26.2 Finding cycles and topological sorting using DFS



DFS can be used to determine whether or not a digraph is a DAG and, if it is, find a topological sort.

- Run DFS on G.
- If G contains a cycle, the traversal will eventually reach a node that points to one that has been seen before. In other words, we will detect a back arc.
- If the traversal finds no back arcs, G is a DAG and the listing nodes in reverse order of DFS `done` times is a topological sort of G.

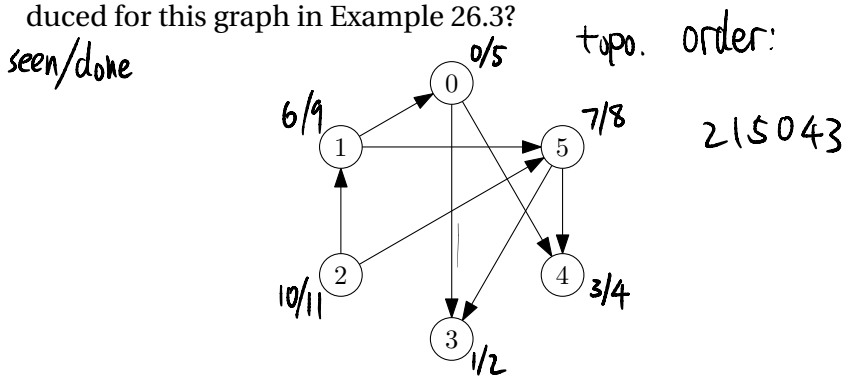
if there's a back-arc, then it's not acyclic

Theorem 26.8. Suppose that DFS is run on a digraph G. Then G is acyclic if and only if there are no back arcs.

Theorem 26.9. Let G be a DAG. Then listing the nodes in reverse order of DFS finishing times yields a topological order of G.

Proof. Consider any arc $(u,v) \in E(G)$. Since G is a DAG, the arc is not a back arc by Theorem 26.8. In the other three cases, Example 24.8 shows that $\texttt{done}[u] > \texttt{done}[v]$, which means u comes before v in the alleged topological order. □

Example 26.10. Find the topological order for the graph shown by running DFS starting at node 0. Is it the same order you produced for this graph in Example 26.3?

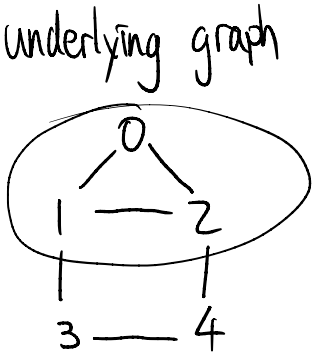


26.3 The girth of a graph

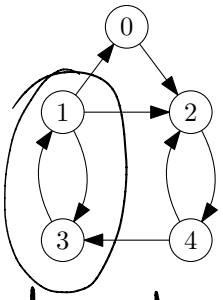
The length of the smallest cycle in a graph is an important quantity. For example, in a communication network, short cycles are often something to be avoided because they can slow down message propagation.

Definition 26.11. The ***girth*** of the graph is the length of the shortest cycle. If the graph has no cycles then the girth is undefined but may be viewed as $+\infty$.

For a digraph we use the term girth for its underlying graph and the (maybe non-standard) term ***directed girth*** for the length of the smallest directed cycle.



Example 26.12. What are girth and directed girth of the following digraph?



girth: 3

directed girth: 2

directed