MATH 566: Lecture 9 (09/22/2026)

Today: * algo for topological ordering
* flow decomposition

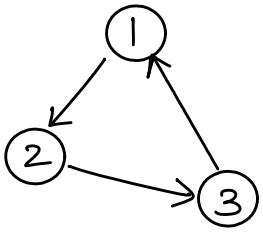
Topological Ordering

Recall the definition from the last lecture...

Def A labeling order(·) is a **topological ordering** if $\forall (i,j) \in A$, we have order(i) < order(j).

We now consider the problem of constructing a topological ordering for a given network, if possible, or certify that a topological ordering does not exist. From that point of view, can we characterize when a network is guaranteed to have a topological ordering?

Not all graphs are guaranteed to have a topological ordering. Directed cycles created obstructions.



No topological ordering is possible

But if G has no directed cycles, we can always find a topological ordering.

In fact, we can prove the following stronger result:

network is acyclic $\iff$ it has a topological ordering.

We outline the results that give one direction of the above equivalence

Lemma If outdegree$(i) \geq 1 \ \forall i \in N$, then the first inadmissible arc of DFS determines a directed cycle.

Corollary 1 If G has no directed cycle, there is at least one node with zero outdegree and at least one node with zero indegree.

Corollary 2 If G has no directed cycle, one can label the nodes so that order$(i) <$ order$(j) \ \forall (i,j) \in A$.

Idea for algorithm Start with nodes that have indegree $= 0$. Assign order $= 1$. Delete these nodes and all arcs going out of these nodes. Adjust node indegrees. Assign order $= 2$ for all nodes with indegree $= 0$ now. Repeat till network is empty.

In practice, we do not actually delete the nodes and arcs — just keeping track of indegrees (and decreasing them as we proceed) will be sufficient.

Pseudocode for topological ordering (from AMO):

```
algorithm topological ordering;
begin
    for all i ∈ N do indegree(i) := 0;
    for all (i, j) ∈ A do indegree(j) := indegree(j) + 1;
    LIST := ∅;
    next := 0;
    for all i ∈ N do
        if indegree(i) = 0 then LIST := LIST ∪ {i};
    while LIST ≠ ∅ do
    begin
        select a node i from LIST and delete it;
        next := next + 1;
        order(i) := next;
        for all (i, j) ∈ A(i) do
        begin
            indegree(j) := indegree(j) - 1;
            if indegree(j) = 0 then LIST := LIST ∪ {j};
        end;
    end;
    if next < n then the network contains a directed cycle
    else the network is acyclic and the array order gives a topological order of nodes;
end;
```

initializing indegrees

counter

different from search!

≡ "deleting" (i, j)

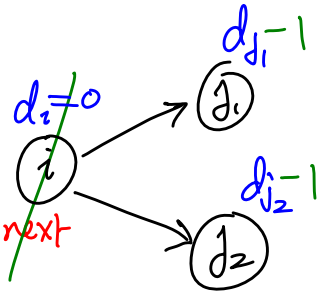


Figure 3.8 Topological ordering algorithm.

Note the similarities to the generic search algorithm. But also note that each node selected from LIST is immediately removed/deleted from LIST here.

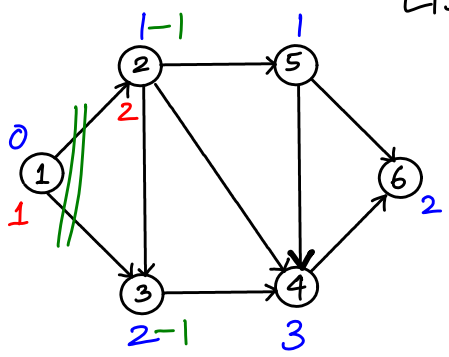
Using similar arguments as used for search, we can see that topological ordering runs in $O(m)$ time.

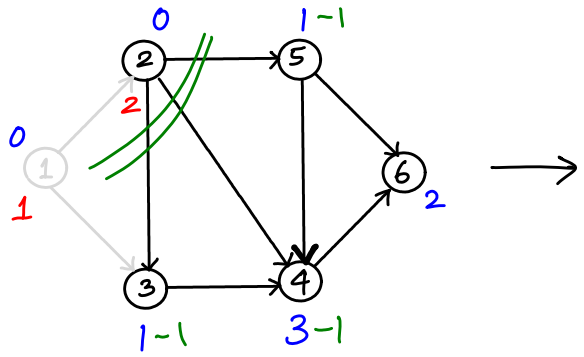
Illustration

We show the steps of topological ordering on a slightly modified network from the one we have seen previously — we include (5,4) in place of (4,5), hence the BFS order(·) is not a topological ordering.

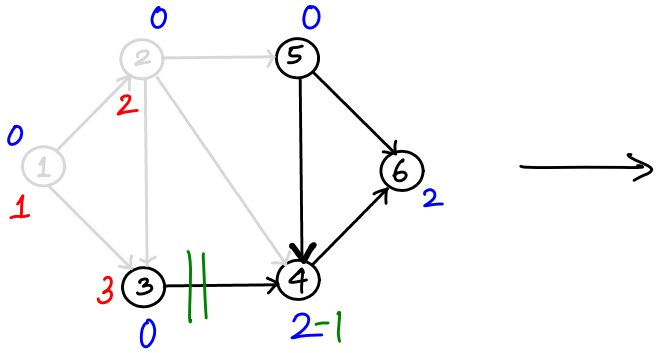
indegree
i
order

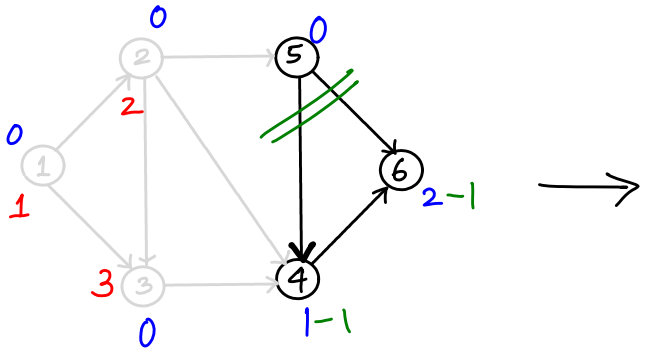
LIST: [1] → [7]



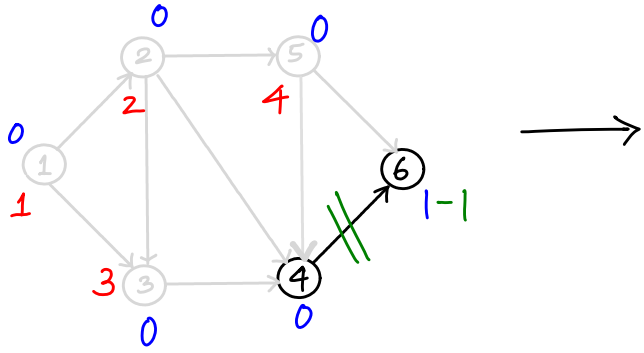


LIST: [3 5] → [5] → [4]

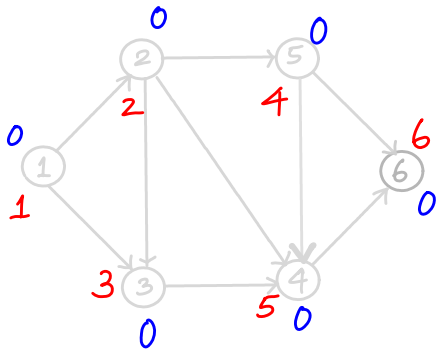




LIST: [4] → [6]

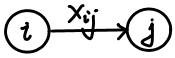


LIST: [6]

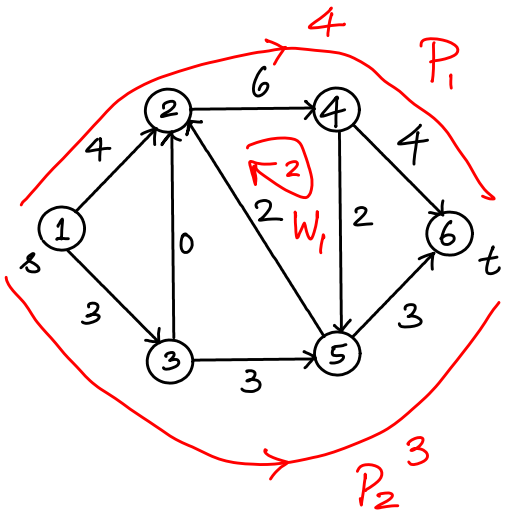


Flow Decomposition

So far, we have considered flow in arcs, using the x_{ij} variables. Alternatively, we could consider flow in paths from supply to demand nodes, and flow in cycles.



Consider this example network, with flows on arcs x_{ij} given. We assume x_{ij} values satisfy flow balance as well as bound constraints.



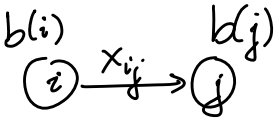
Instead, we could consider flows along two paths and a cycle.

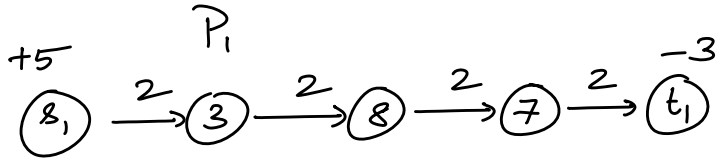
P_1: 1-2-4-6 sending 4 units

P_2: 1-3-5-6 sending 3 units

W_1: 2-4-5-2 circulating 2 units.

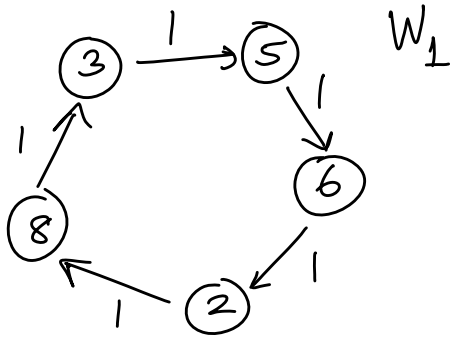
Let P_1 be a path from s_1 to t_1 as shown here:





We specify that the intermediate nodes in P_1 conserve flow, i.e., inflow = outflow is satisfied when restricted to the path. Note that each node and arc in P_1 could be part of other paths and/or cycles. In particular, the value $b(s_1) = 5$ is not determined just by the flows in the arcs in P_1.

To describe flow along a cycle W_1, we specify that every node in W_1 satisfies inflow = outflow.



We denote by $f(P)$ and $f(W)$ the flow in path P and cycle W.
We also specify paths and cycles using indicator variables δ_{ij} for each arc (i,j).

$$\delta_{ij}(P) = \begin{cases} 1, & \text{if } (i,j) \in P \\ 0, & \text{otherwise.} \end{cases} \qquad \delta_{ij}(W) = \begin{cases} 1, & \text{if } (i,j) \in W \\ 0 & \text{otherwise} \end{cases}$$

Claim Given flows in a set of paths $\mathcal{P}$ and set of cycles $\mathcal{W}$, we can get the flows in arcs using the δ_{ij} indicators:

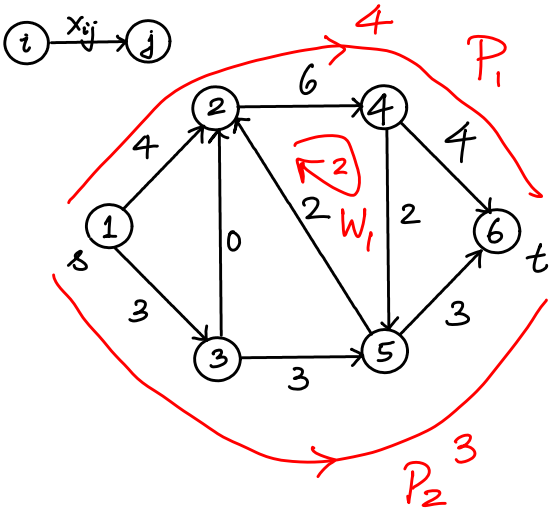
$$x_{ij} = \sum_{P \in \mathcal{P}} f(P)\delta_{ij}(P) + \sum_{W \in \mathcal{W}} f(W)\delta_{ij}(W) \quad \forall (i,j) \in A.$$

Consider the example again. Here, $f(P_1) = 4$, $f(P_2) = 3$, and $f(W_1) = 2$.

Arc $(2,4)$ is part of $P_1 = 1\text{-}2\text{-}4\text{-}6$ and $W_1 = 2\text{-}4\text{-}5\text{-}2$.

Indeed, we get

$$x_{24} = f(P_1) + f(W_1) = 4 + 2 = 6.$$



The more interesting question is, given a set of arc flows (x_{ij}'s), can you find an equivalent collection of path and cycle flows?

Yes! We repeatedly search for paths/cycles, and send flows along them until all arc flows are exhausted.

Flow decomposition algorithm

We work with intermediate flows y_{ij}. We start with $y_{ij} = x_{ij}$. We then search to find a path P from a supply node to a demand node, or a cycle W. We then push flow along P or W, update y_{ij}'s, $b(i)$'s, as well as the associated network. We repeat till $y_{ij} = 0 \ \forall (i,j) \in A$ and $b(i) = 0 \ \forall i \in N$.

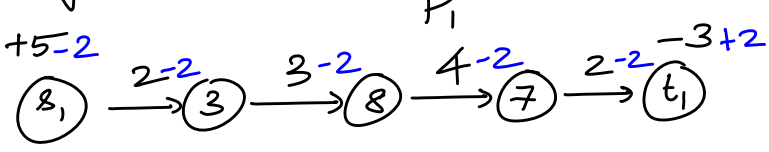
How much can we push? Consider the path $P = s - i_1 \cdots i_r - t$.

We define $\Delta(P) = \min\{b(s), -b(t), y_{ij} \ \forall (i,j) \in P\}$, and set $f(P) = \Delta(P)$.

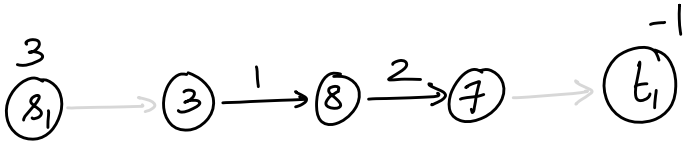
($\Delta(P)$ → "capacity" of P)

e.g., let P_1 be

P_1



$\Delta(P_1) = \min\{5, 3, 2, 3, 4, 2\} = 2.$



Set $f(P_1) = \Delta(P_1) = 2$.

After the update (to y_{ij}, $b(i)$), the arcs $(s_1, 3)$ and $(7, t_1)$ are no longer considered in the network.