MATH 566: Lecture 9 (09/17/2024)

Today: * algo for topological ordering
* flow decomposition

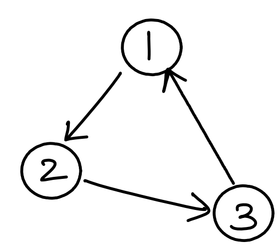
Topological Ordering

Recall the definition from the last lecture...

Def A labeling order$(\cdot)$ 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) $\forall (i,j) \in A$.

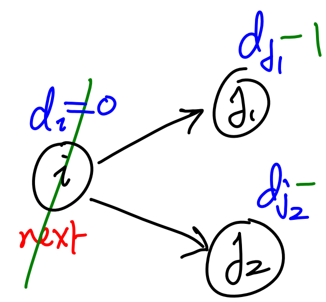
Idea for algorithm Start with nodes that have indegree $= 0$. Assign order$=1$. Delete these nodes and all arcs going out of these nodes. Adjust node indegrees. Assign order$=2$ for all nodes with indegree$=0$ now. Repeat till network is empty.

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;   ← we assume indegrees are not known
    LIST : = Ø;
    next : = 0;   ← counter
    for all i ∈ N do
        if indegree(i) = 0 then LIST : = LIST ∪ {i};
    while LIST ≠ Ø do
    begin
        select a node i from LIST and delete it;   → different from search!
        next : = next + 1;
        order(i) : = next;
        for all (i, j) ∈ A(i) do
        begin
            indegree(j) : = indegree(j) − 1;   ≡ "deleting" (i,j)
            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;
```

Figure 3.8 Topological ordering algorithm.



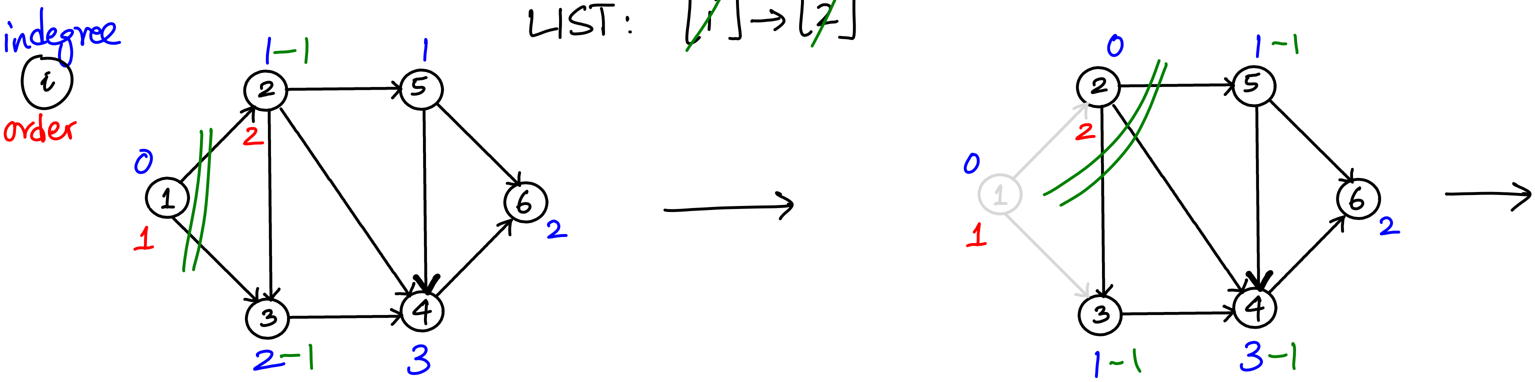
Note the similarities to the generic search algorithm. But also note that each node selected from LIST is immediately removed/deleted from LIST here.

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

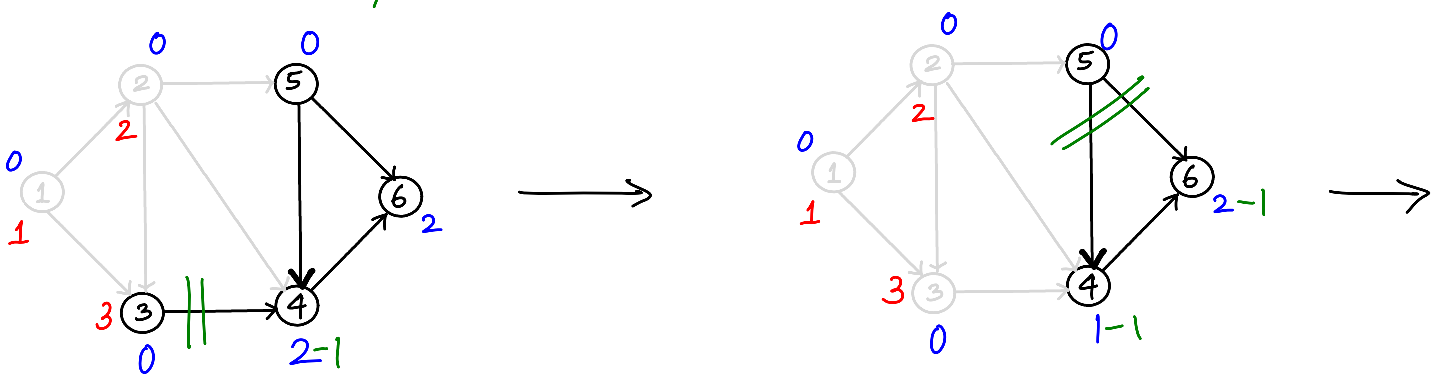
Illustration

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.

LIST: $[\cancel{1}] \to [\cancel{2}]$

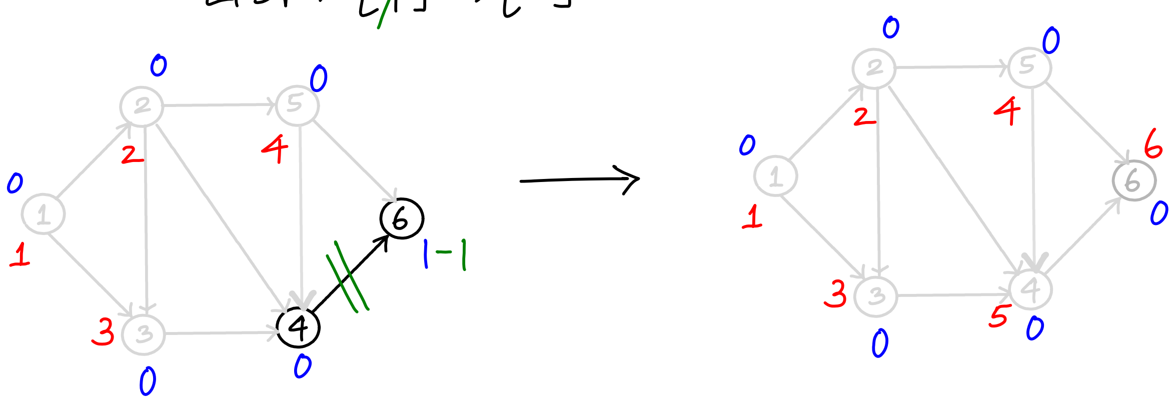


LIST: $[\cancel{3}\ 5] \longrightarrow [\cancel{5}] \longrightarrow [4]$



LIST: $[\cancel{4}] \to [6]$

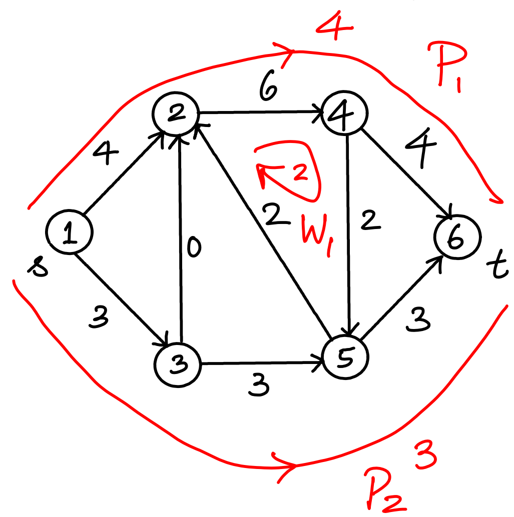
LIST: $[\cancel{6}]$



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.

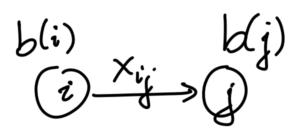


Instead, we could consider flows along two paths and a cycle.

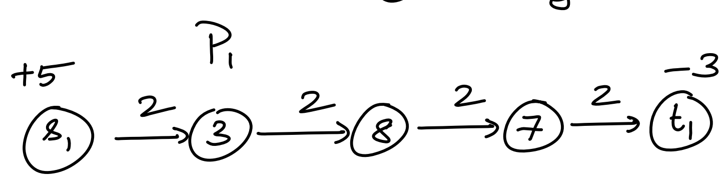
P_1 : 1-2-4-6 sending 4 units

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

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

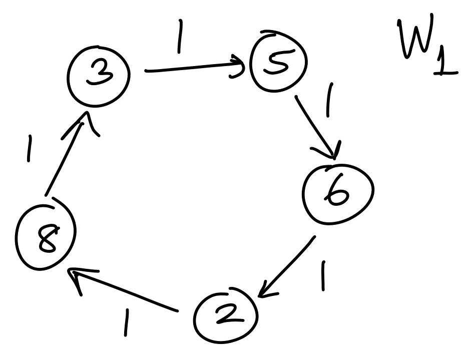


Let P_1 be a path from s_1 to t_1 as shown here:



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:

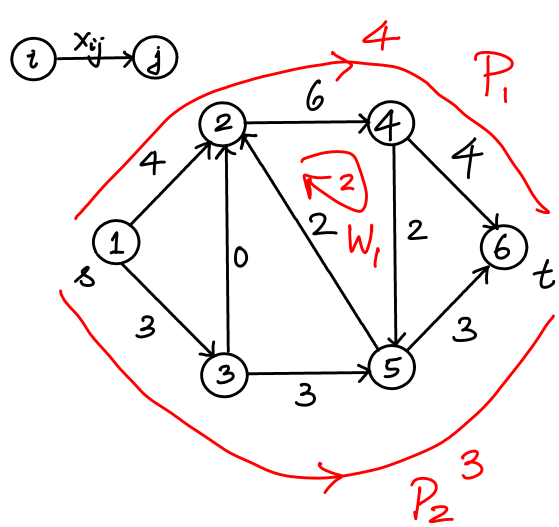
$$x_{ij} = \sum_{P \in \mathcal{P}} f(P)\delta_{ij}(P) + \sum_{W \in \mathcal{W}} f(W)\,\delta_{ij}(W) \quad \forall (i,j) \in A.$$

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 $b(i) = 0 \ \forall i \in N$.

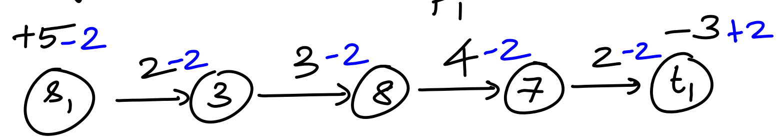
How much can we push? Consider the path $P = s - i_1 \cdots i_r - t$.

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

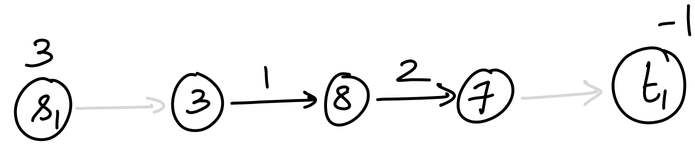
$\Delta(P)$ → "capacity" of P

P_1

e.g., let P_1 be



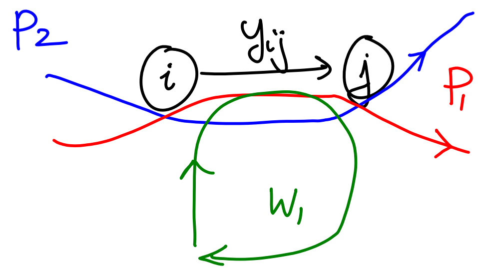
$\Delta(P_1) = \min\{5, 3, 2, 3, 4, 2\} = 2.$



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.

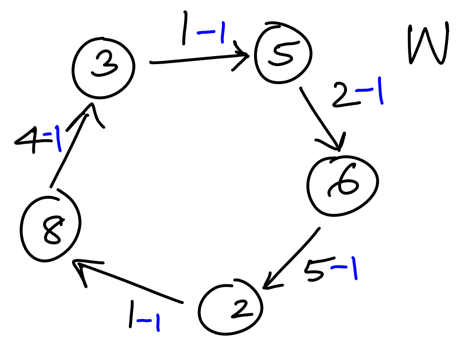
Note that (i,j) can be part of multiple paths and/or cycles, but it is always a forward arc in each path and cycle.



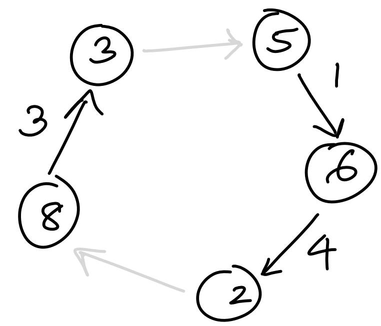
For a cycle W, we set $\Delta(W) = \min\{y_{ij} \mid (i,j) \in W\}$.
↳ capacity of cycle W

Here, $\Delta(W_1) = 1$.

We set $f(W_1) = \Delta(W_1) = 1$.



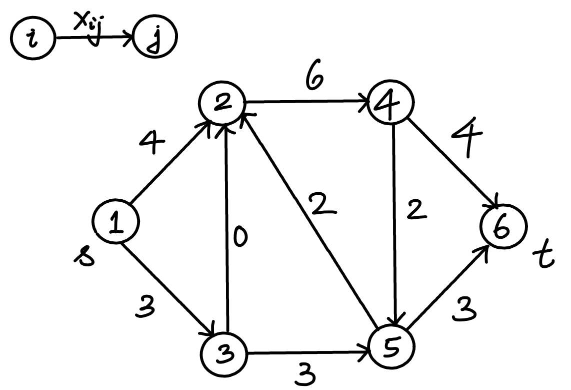
After update, $(3,5)$ and $(2,8)$ are removed from further consideration.

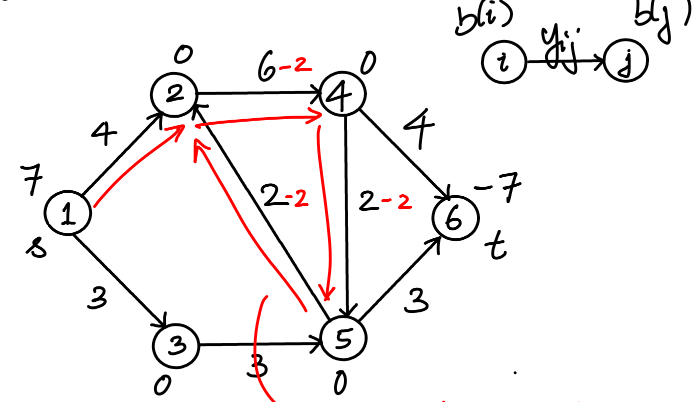


We run DFS repeatedly. If we find a directed cycle, we push its capacity of flow. Else if we find a path from a supply to a demand node, we push its capacity. We update the intermediate flows and the network, and continue. See the algorithm in the handout posted on the course web page.

Illustration

Consider the input flow. We start by initializing $b(i)$ and y_{ij} values using x_{ij} values.





inadmissible arc.
identifies $W_1 = 2\text{-}4\text{-}5\text{-}2$

Notice we do not have (3,2) in the starting network, as $y_{32} = 0$ to start with.

We maintain sets of supply and demand nodes. $\mathcal{S}$, $\mathcal{D}$.

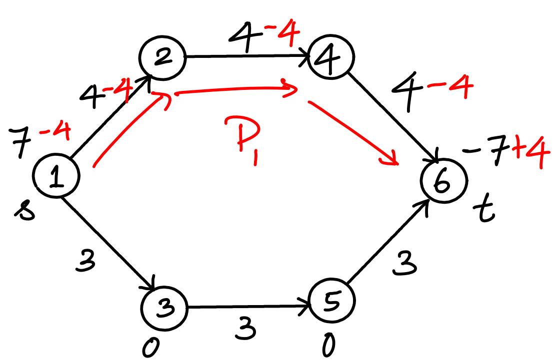
$\mathcal{S} = [1]$, $\mathcal{D} = [6]$ at the start.

Start with $s = 1$ (taken from $\mathcal{S}$).

We perform DFS. We identify cycle $W_1 = 2\text{-}4\text{-}5\text{-}2$. We set

$\mathcal{W} = \{W_1\}$. $\Delta(W_1) = \min\{6, 2, 2\} = 2$. Set $f(W_1) = 2$.

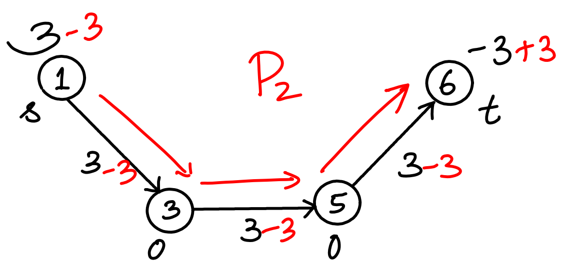
The updated network is shown here. We still have $\mathcal{S} = [1]$, $\mathcal{D} = [6]$.



Starting with $s = 1$, we do DFS. We identify $P_1 = 1\text{-}2\text{-}4\text{-}6$, with $\Delta(P_1) = \min\{7, 4, 4, 4, 7\} = 4$.

We set $\mathcal{P} = \{P_1\}$, $f(P_1) = 4$.

$S = [1]$, $\mathcal{D} = [6]$



Start with $s=1$, do DFS. We find $P_2 = 1\text{-}3\text{-}5\text{-}6$ with

$$\Delta(P_2) = \min\{3, 3, 3, 3, 3\} = 3.$$

We set $\mathcal{P} = \mathcal{P} \cup \{P_2\} = \{P_1, P_2\}$, and $f(P_2) = 3$.

After this iteration, all $b(i) = 0$, $y_{ij} = 0$. Thus we have decomposed the arc flow into flows along paths P_1, P_2, and cycle W_1.

We are trying to account for all y_{ij} and $b(i)$ for supply/demand nodes, by assigning amounts out of y_{ij} and $b(i)$ into $f(P)$ and $f(W)$. Ultimately, we want to get $b(i) = 0 \ \forall i$, and $y_{ij} = 0 \ \forall (i,j)$, at which point, $f(P)$ and $f(W)$ for $P \in \mathcal{P}$ and $W \in \mathcal{W}$ account for all flows.

The input flow $\bar{x} = [x_{ij}]$ (vector of x_{ij} values) is assumed to satisfy flow balance, and bounds. But it need not necessarily be an optimal flow — notice that we do not worry about costs in this decomposition.

Similarly, the bounds (u_{ij}) do not play a direct part in the flow decomposition. Naturally, $f(P)$ and $f(W)$ cannot exceed y_{ij}, and hence x_{ij} values.