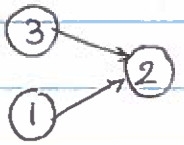
10/4/08

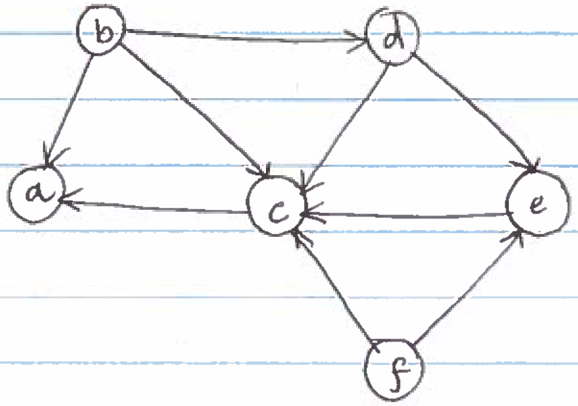
TOPOLOGICAL SORT

- Set of tasks
- Precedence constraints: A set of pairs eg. (v_i, v_j), meanin task v_i must occur before task v_j.
- Can be represented as a directed graph.



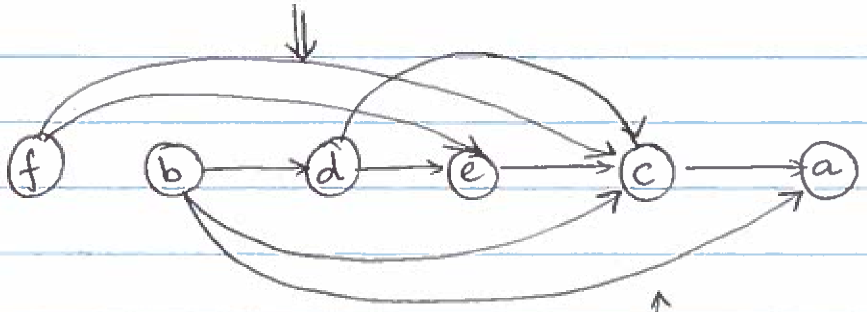
3 must occur before 2
and
1 must occur before 2.

- Use case: Course pre-requisities, dependency installation



Define: a topological order of digraph is ordering of its nodes as $v_1, v_2, \ldots, v_n$ so that for every edge (v_i, v_j) we have $i<j$.

⇓



All edges from left to ri

This is a topological order.

? How do we know a topological order exists or not

Recall

DAG = directed acyclic graph

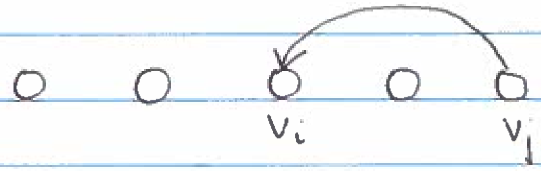
→ <u>Theorem</u>: G has a topological order iff G is a DAG

<u>Proof</u>:

($\Rightarrow$) By contradiction: Suppose G has a cycle and has a topological order.

Let v_i be the $\underbrace{\text{first node}}_{\text{Left-most}}$ in the ordering that's in the cycle.

Let v_j be the node just before v_i in the cycle.



$E(v_j, v_i)$ is out of order!

f by algorithm!

($\Leftarrow$) Proof by topological sort algorithm:

Given DAG, returns topological order of G.

? How would you do it for courses with pre-requisite

-Start with course with no pre-requisite.

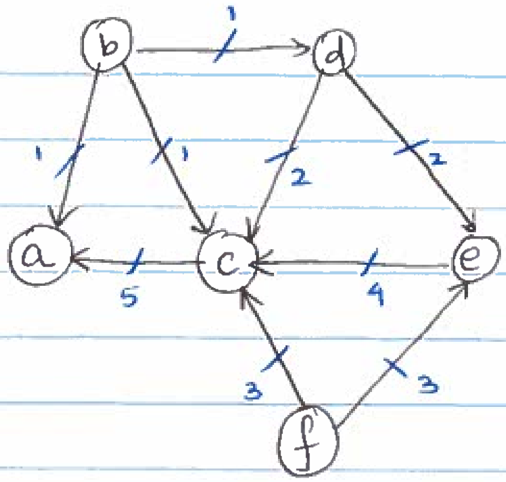
SAME IDEA APPLIED!

want to do this in $\Theta(1)$

$\Theta(n)$ Find a node v with no incoming edges and order it first

nes

Recursively compute topological ordering of $G \setminus \{v\}$ (Delete v and all its edges)



1	2	3	4	5	6
b	d	f	e	c	a

Note: Get different topologi
order depending on your ch

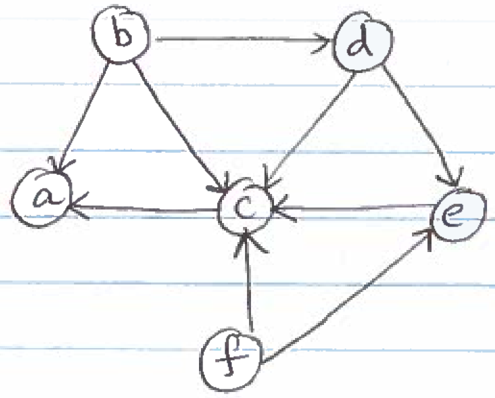
Running time of algorithm?

→ $\Theta(n+m)$ implementation of topological sort :

Keep array in-count storing the # of incoming edges to every node.

Keep linked list S = set of remaining nodes with no incoming edges. [n = # nodes, m = # ed

Initialization : $\Theta(n+m)$



	a	b	c	d	e	f
in_count	2	0	4	1	2	0
	1	Ø	3	0	2	0
	1	Ø	2	0	1	Ø
	1	Ø	1	Ø	0	Ø
	1	Ø	0	Ø	Ø	Ø
	0	Ø	Ø	Ø	Ø	Ø

S = { b, f, d, e, c, a }

Updates : ★ remove first v from S

★ decrement in-count[w] for all edges (v,
and add w to S if in-count[w] hits 0

Constant work per node!

→ <u>lemma</u>: If G is a DAG, then G has a node with no incoming edges.

<u>Proof</u>: Suppose every node has an incoming edge. Start at any node and follow edges backward from it.

Since finite number of nodes, must visit same node twice. Between consecutive visits of this node there is a cycle.

<u>QED</u>.

→ <u>Proof of Corectness</u> of topological sort algorithm:

- Prove by induction.

ALWAYS REMEMBER TO WRITE WHAT THE INDUCTION IS ON

- <u>Induction</u> on the $\underbrace{\text{\# of connected nodes}}_{n}$.

<u>Base case</u>: $n=1$ ⇒ Holds true

<u>Ind. hypothesis</u>: Assume algorithm gives valid top. order for n nodes.

<u>Want to Prove</u>: Algo. gives valid top. order for $n+1$ nodes.

- In the algo, $\underbrace{G \setminus \{v\}}_{n \text{ nodes}}$ is still a DAG if G was.

(Inductive assumption applies)

So By inductive assumption, recursive call gives a top. order on $G \setminus \{v\}$

Add to left (edges going out)

- Adding v first is still a topological sort since it has no incoming edges.

QED.

GREEDY ALGORITHMS

- Class of algorithms that have a certain flavour.
- Gayle-Shapley algorithm can be seen as a greedy algorithm.

⇒ **INTERVAL SCHEDULING:** (Refer Slides)

s_j = Start time of job j

f_j = Finish time of job j.

- Sort jobs by finish times → go through them in order and choose the ones that ~~all~~ are compatible with previous ones.

- Runtime of algorithm:

Sorting: $\Theta(n\log n)$

n times [$\Theta(1)$

∴ Algorithm is $\Theta(n\log n)$

To check that job j compatible with A:

$\Theta(1)$ {
- Remember job j^* that was most recent.
- job is compatible w A if $s_j \geq f_{j^*}$