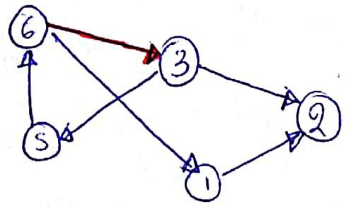
EE360C ALGORITHMS - FALL 2018 - 4 October - 11-12:30

Topological Sort

Set of tasks

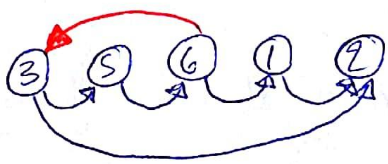
Precedence constraints: task v_i must occur before v_j



3 before 2

and

1 before 2

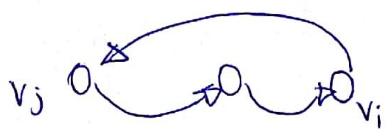


<u>Def</u> Topological order of a digraph is an ordering of its nodes as $v_1, v_2, \cdots, v_n$ s.t. for every edge (v_i, v_j) we have $i<j$.

<u>Lemma</u>: If cycle then no topological order.

<u>Pf</u>: By contradiction, assume it has topological order & a cycle. Let v_j be the first node in the top. order that's in the cycle. Let v_i be the node right before it in the cycle. Then edge(v_i, v_j) is out of order: $i > j$!

$\square$



Recall: DAG: Directed Acyclic graph

Theorem: G has a topological order iff G is a DAG.

($\Rightarrow$) already proved by contrapositive (lemma 8)

($\Leftarrow$)

Topological sort algorithm: Given DAG G, return some topological order of G.

Find a node v with no incoming edges and order it first.

Recursively compute top. order of $\underbrace{G\setminus\{v\}}_{\text{delete } v \text{ and all edges involving } v}$

and append this order after v.

Lemma: If G is a DAG, then G has a node with no incoming edges.

Pf: By contradiction. suppose every node has an incoming edge. Start at any node and follow edges backward from it. Since we have a finite # of nodes, we must visit the same nod twice. Between successive visits we have a cycle.

QED.

$\boxed{\text{Proof of corectness of top. sort. alg.}}$:

By induction on the # of remaining nodes. $\boxed{\text{G is a DAG}}$

Base case: n=1

Ind. hypothesis: Assume alg. gives top. order for n nodes.
Want to prove that it gives topological order for n+1 nodes.

- $G \setminus \{v\}$ is still a DAG, now with n nodes. So by inductive hypothesis, recursive call gives a topological order on $G \setminus \{v\}$.
  - Placing v first is still a top. order since v only has outgoig edges. $\square$

$\Theta(n+m)$ implementation of top. sort.:

Keep array in_count[n] = # of incoming edges to u.
Keep linked list S = set of remaining nodes with no incoming edges.
Initialization: $\Theta(n+m)$ via a single scan through the graph.
Updates:
- remove first node v from S.
- decrement in_count[w] for all edges (v,w) and add w to S if in_count[w] hits 0. } Θ(# of edges out of v)

INTERVAL SCHEDULING

Complexity of greedy algorithm:

→ $\Theta(n \log n)$ to sort jobs by earliest finishing time

→ n repetitions of the loop ($j=1$ to n)
perform an $\Theta(1)$ comparison
(let j^* be the job most recently added to A. we check to see if $s_j \geq f_{j^*}$)

$\Theta(n \log n)$ total