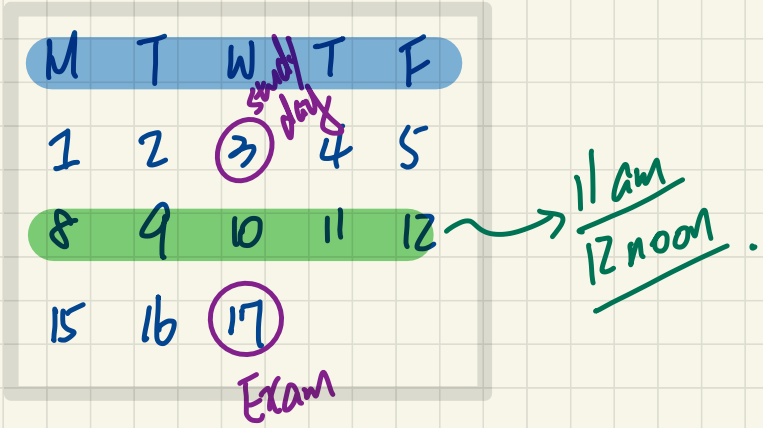
Lecture 20 – Nov 26

Graphs

Directed Acyclic Graphs (DAGs)
Topological Ordering
Topo. Sort: Time Complexity, Tracing
Topo. Sort: Sequentializing Updates

Announcements/Reminders

- Today's class: notes template posted
- One in-person make-up lecture
- One or two exam review sessions



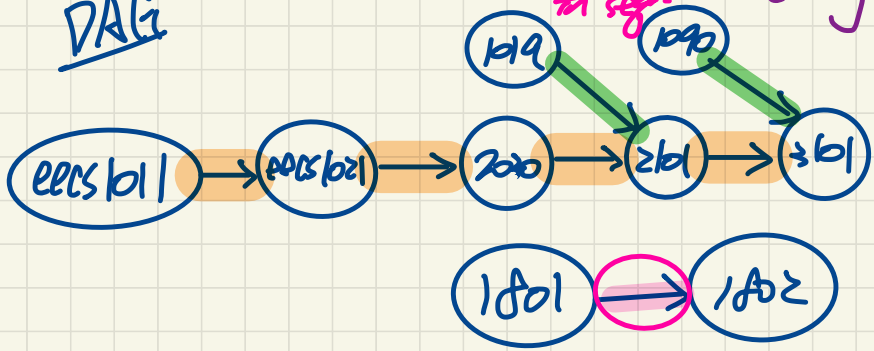
Directed Acyclic Graphs (DAGs)

$\forall i, j \cdot 1 \le i, j \le n \wedge (v_i, v_j) \in E \Rightarrow i < j$

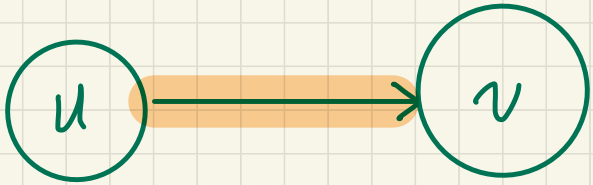
DAG $G = (V, E)$ → dependency

- DAG
  - ↳ $v \rightarrow u$
  - ↳ no cycles.

DAG

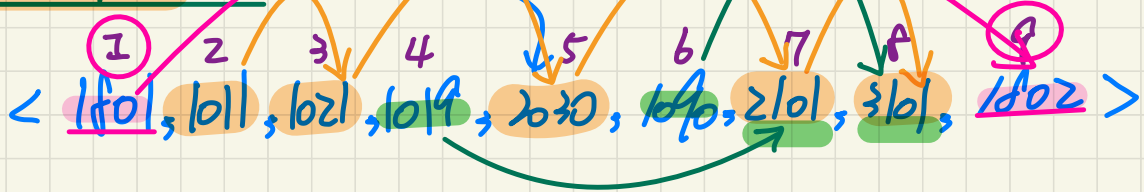


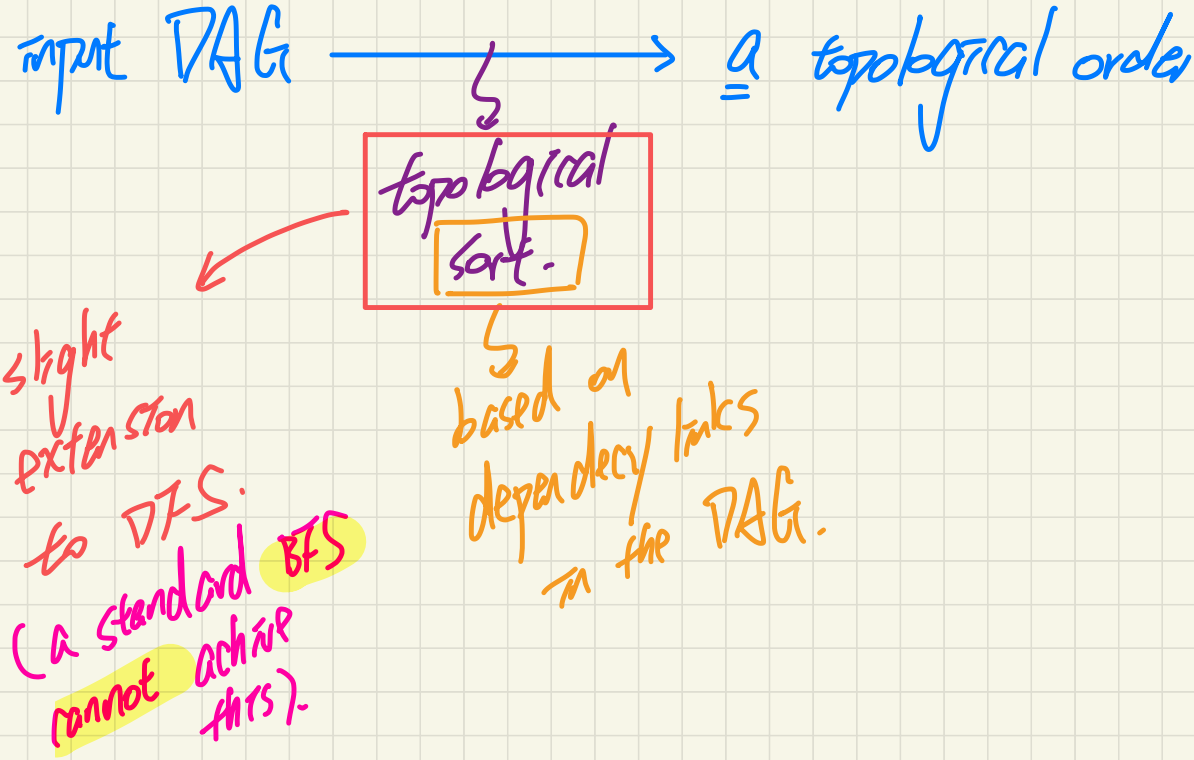
Exercise : DAG vs. Tree



u must occur before v

Topological Order (sequentialized DAG)
↳ not unique.




input DAG
a topological order
topological sort.
slight extension to DFS.
(a standard BFS cannot achieve this).
based on dependency links in the DAG.

Topological Sort on a DAG: Time Complexity

```
Iterable<Vertex<V>> topologicalSort(Graph<V, E> g) {
  ArrayList<Vertex<V>> order = new ArrayList<>();
  for(Vertex<V> v: g.vertices()) {
    if(!v.isVisited()) {
      DFStopo(g, v, order)
    }
  }
  return order;
}
```

assumption: each vertex is marked as "visited" or "unvisited".

topological order so far.

```
1  DFStopo(Graph<V, E> g, Vertex<V> v, ArrayList<Vertex<V>> order) {
2    Stack s = new LinkedStack(); v.setVisited(); s.push(v);
3    while(!s.isEmpty()) {
4      Vertex<V> top = s.peek();
5      Iterator<Edge<E, V>> it = g.outGoingEdges(top);
6      boolean foundUnexploredEdge = false;
7      while(it.hasNext() && !foundUnexploredEdge) {
8        Edge<E, V> e = it.next();
9        Vertex<V> opposite = e.getDestination();
10       if(!opposite.isVisited()) { /* discovery edge */
11         foundUnexploredEdge = true;
12         opposite.setVisited(); s.push(opposite);
13       }
14     }
15     if(!foundUnexploredEdge) { order.addFirst(top); s.pop();}
16   }
17 }
```

visit all incident edges once.

each edge is marked only once.

each run only "sorts" vertices in some connected component.

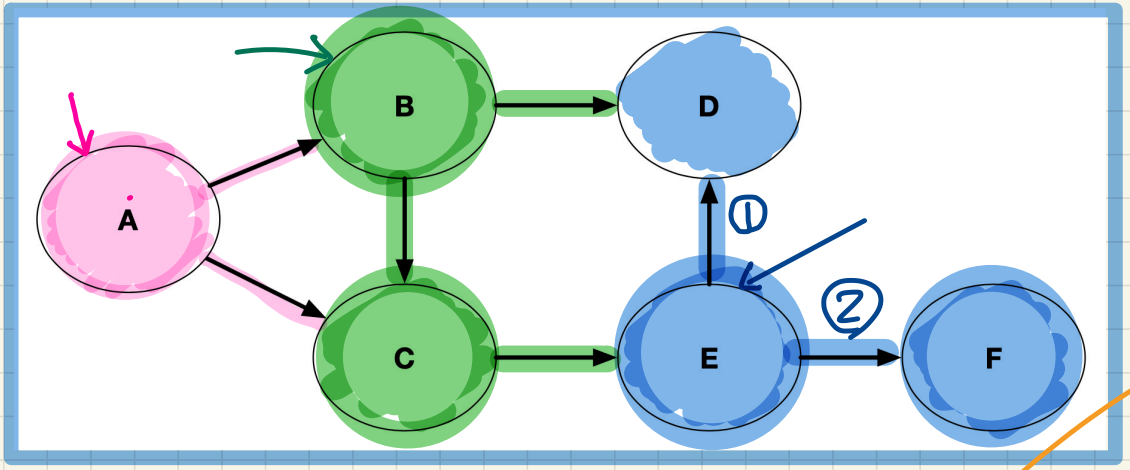
every vertex is pushed to and popped from the stack once.

RT: $O(|V| + |E|) = O(n + m)$

topological order is reverse of vertices' finishing time.

Topological Sort on a DAG: Example (1)

Assume T.E. visited in alphabetic order.



Iteration 1 DFStopo(E, <>)

order: < E F D >

push	pop
E	D
D	F
F	E

Iteration 2 DFStopo(B, <E, F, D>)

B C E F D

push	pop
B	C
C	B

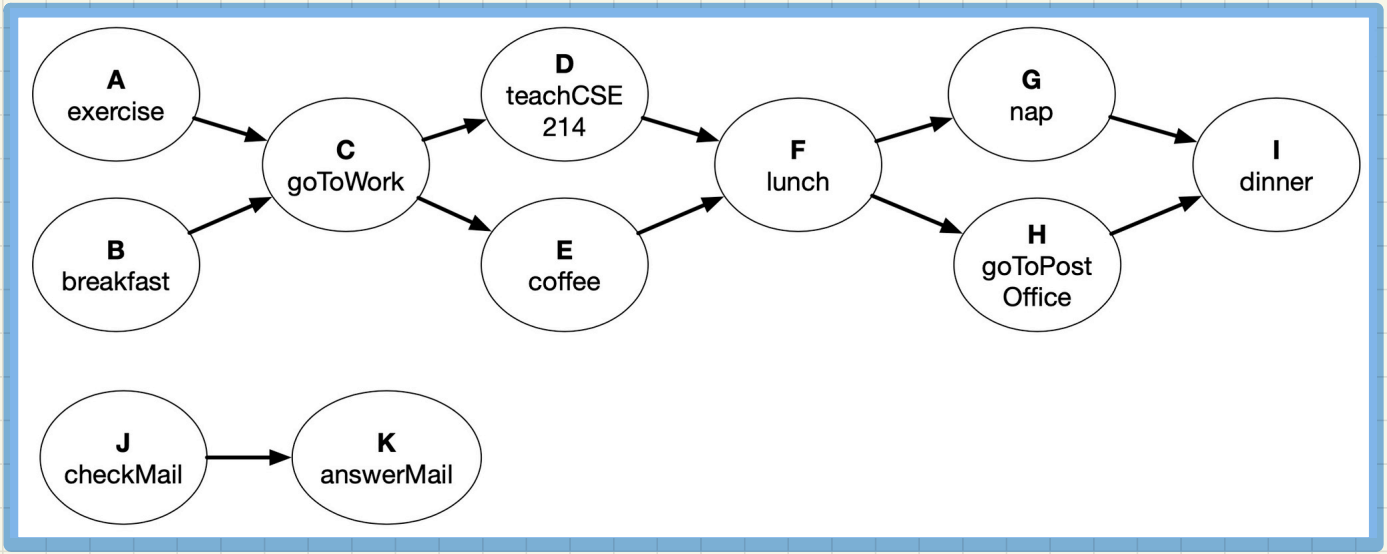
Iteration 3. DFStopo(A, <B, C, E, F, D>)

push	pop
A	A

A B C E F D

ultimate topological order to return

Topological Sort on a DAG: Example (2)



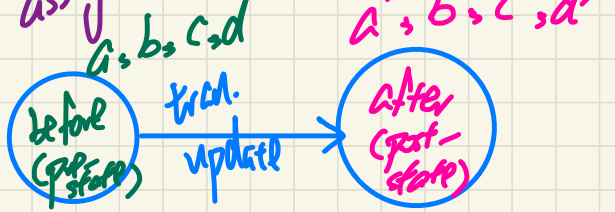
Topological Sort on a DAG: Example (3)

$$
\begin{aligned}
& c' = b' - a' + d \\
\wedge\ & b' = a' * d' \\
\wedge\ & d' = a + c + 5 \\
\wedge\ & a' = a + d'
\end{aligned}
$$

post-state values of vars.

* $c' = b'$
↳ b' should be computed before c'

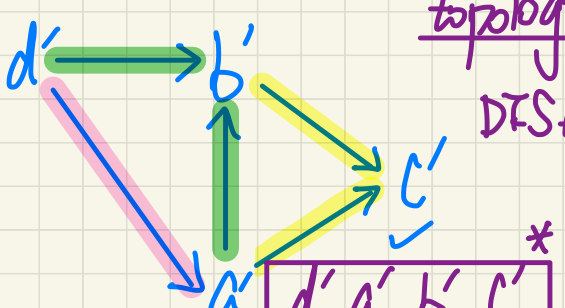
↝ specification of transactional updates (as constraints, not var. assignments)?



Task: Sequentialize the specified transactional updates.

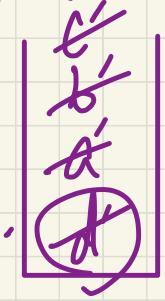
```
old_a := a
old_b := b
old_c := c
old_d := d
d := old_a + old_c + 5
a := old_a + d
b := a * d
c := b - a + old_d
```

DAG



topological sort

$DFS_{topo}(d', \langle\rangle)$



push	pop
d'	c'
a'	b'
b'	a'
c'	d'

* $d'\ a'\ b'\ c'$
topological order.