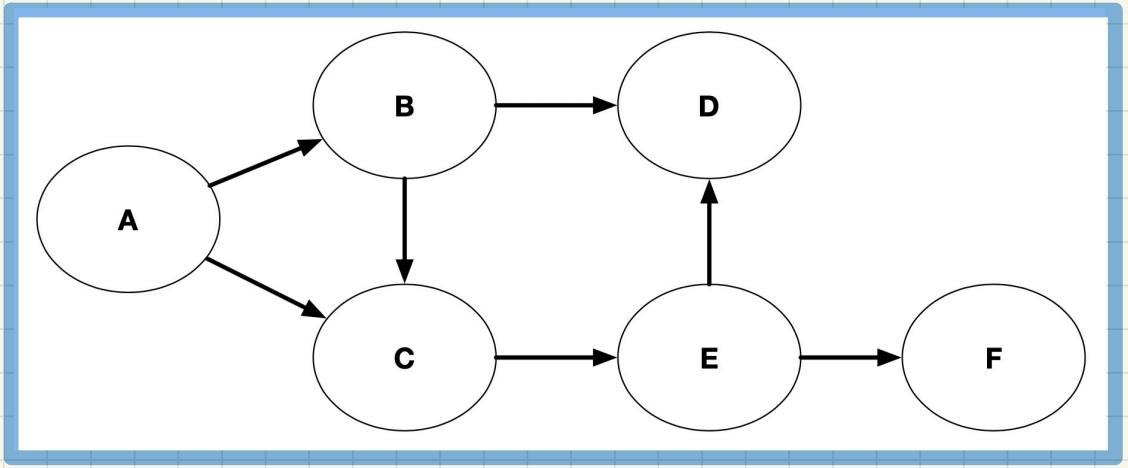
Directed Acyclic Graphs (DAGs)

Topological Sort on a DAG: Time Complexity

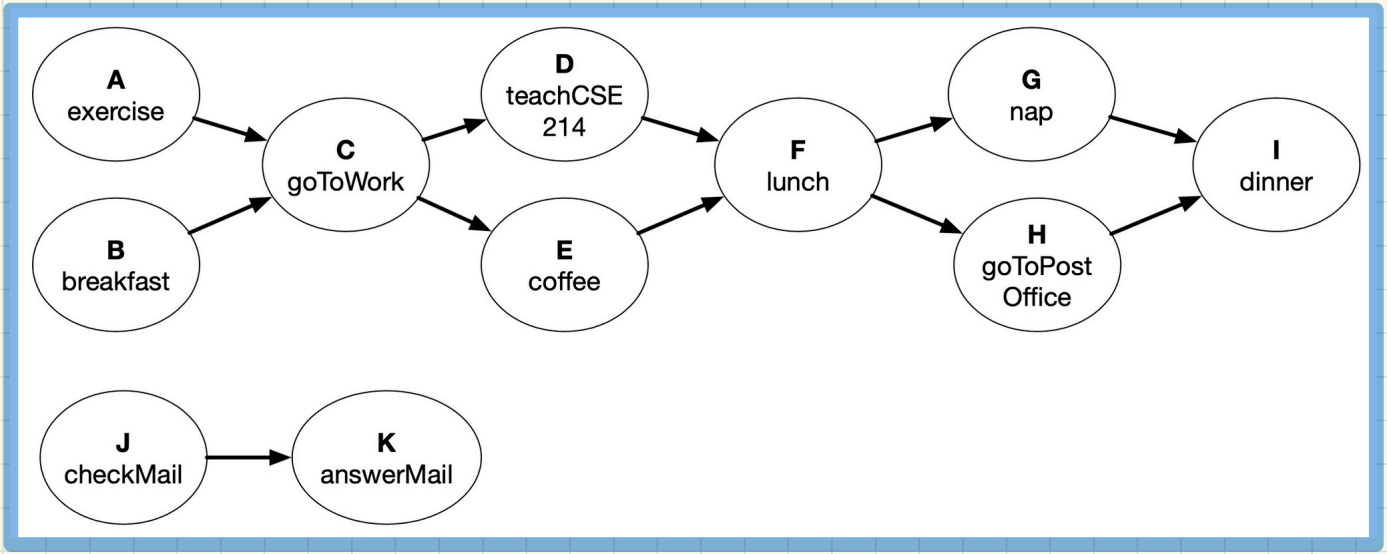
```
Iterable<Vertex<V>> topologicalSort(Graph<V, E> g) {
  ArrayList<Vertex<V>> order = new ArrayList<>();
  for(Vertex<V> v: g.vertices()) {
    if(!v.isVisited()) {
      DFStopo(g, v, order)
    }
  }
  return order;
}
```

```
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 }
```

Topological Sort on a DAG: Example (1)



Topological Sort on a DAG: Example (2)



Topological Sort on a DAG: Example (3)

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

Task: Sequentialize the specified transactional updates.