Basic Graph Algorithms

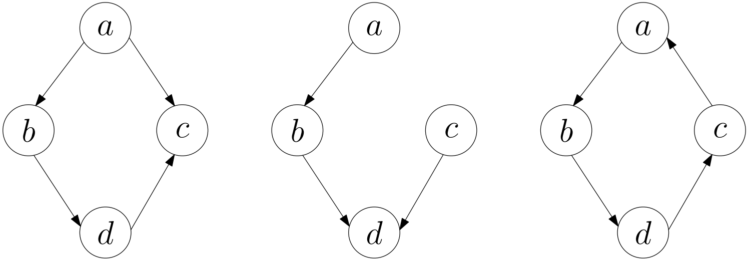
- Exploring Graphs
- Depth First Search
- DFS Forest - Start and Finish Time
- DAG, Topological Sorting
- Strongly Connected Components
- Breadth First Search
- Bipartite Graphs

IMDAD ULLAH KHAN

Directed Acyclic Graphs (DAG)

A DAG is a directed graph that contains no directed cycle

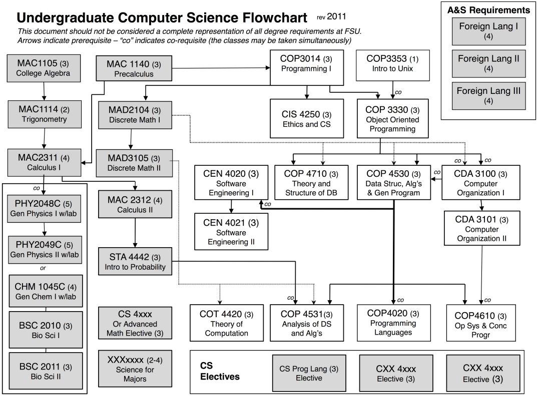
Which ones of the following are DAGs?



Directed Acyclic Graphs (DAG)

A DAG is a directed graph that contains no directed cycle

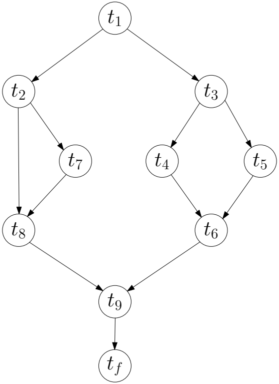
- Model precedence constraints (e.g. course pre-requisites)



Directed Acyclic Graphs (DAG)

A DAG is a directed graph that contains no directed cycle

- Model dependency constraints (e.g. package dependencies)



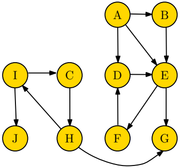
Directed Acyclic Graphs (DAG)

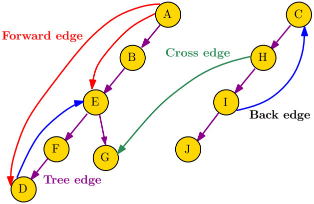
A DAG is a directed graph that contains no directed cycle

- Clearly precedence or dependency graphs cannot have directed cycles
- Could have undirected cycles

DFS Forest: Types of Edges

Overlay all edges of a digraph G onto its DFS forest





- Tree edge - Edge used in the DFS (parent to child)
- Back edge - Edge from a node to a non-parent ancestor
- Forward edge - Edge from a node to a non-child descendant
- Cross edge - Edge from a node in one tree to a node in another

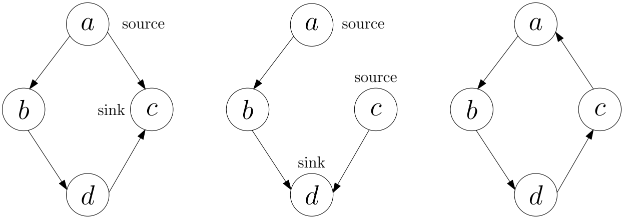
Lemma: A digraph G has a directed cycle if and only if the DFS forest of G has a back edge

DAG: Source and Sink

A DAG is a directed graph that contains no directed cycle

source: A node v in a digraph is a source , if $deg^{-}(v) = 0$

sink: A node v in a digraph is a sink, if $deg^{+}(v) = 0$



DAG: Source and Sink

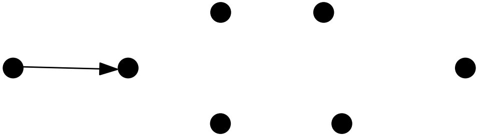
A DAG is a directed graph that contains no directed cycle

source: A node v in a digraph is a source , if $deg^{-}(v) = 0$

sink: A node v in a digraph is a sink, if $deg^{+}(v) = 0$

Every DAG has a source and a sink

If no sink exists, then every vertex has out-degree ≥ 1
Start from a vertex, in each step take an unused outgoing edge



DAG: Source and Sink

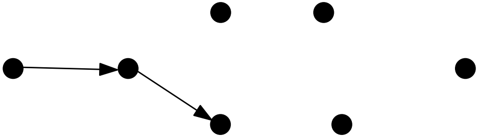
A DAG is a directed graph that contains no directed cycle

source: A node v in a digraph is a source , if $deg^{-}(v) = 0$

sink: A node v in a digraph is a sink, if $deg^{+}(v) = 0$

Every DAG has a source and a sink

If no sink exists, then every vertex has out-degree ≥ 1
Start from a vertex, in each step take an unused outgoing edge



DAG: Source and Sink

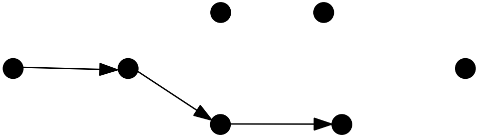
A DAG is a directed graph that contains no directed cycle

source: A node v in a digraph is a source , if $deg^{-}(v) = 0$

sink: A node v in a digraph is a sink, if $deg^{+}(v) = 0$

Every DAG has a source and a sink

If no sink exists, then every vertex has out-degree ≥ 1
Start from a vertex, in each step take an unused outgoing edge



DAG: Source and Sink

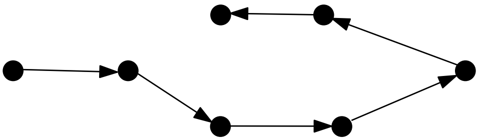
A DAG is a directed graph that contains no directed cycle

source: A node v in a digraph is a source , if $deg^{-}(v) = 0$

sink: A node v in a digraph is a sink, if $deg^{+}(v) = 0$

Every DAG has a source and a sink

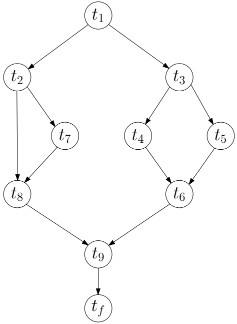
If no sink exists, then every vertex has out-degree ≥ 1
Start from a vertex, in each step take an unused outgoing edge

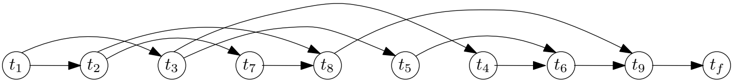


cannot get stuck unless a cycle exists

Topological Order

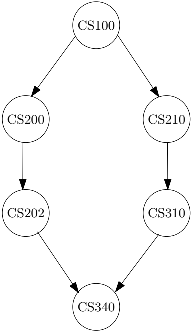
A topological order of a digraph $G = (V, E)$ is an ordering $v_1, v_2, \ldots, v_n$ of V for every edge (v_i, v_j) we have $i < j$

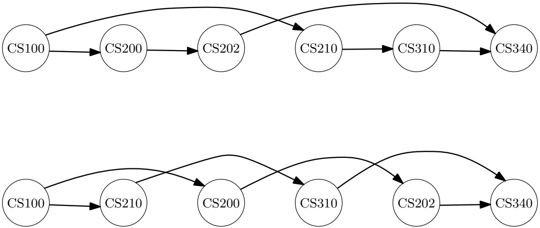




Topological Order

A topological order of a digraph $G = (V, E)$ is an ordering $v_1, v_2, \ldots, v_n$ of V for every edge (v_i, v_j) we have $i < j$





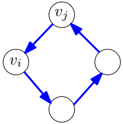
Topological order needs not to be unque

Topological Order

If G has a cycle, then no hope for a topological ordering

If G has a topological ordering, then G is a DAG

- Suppose G has a topological order and G has a cycle C
- Let v_i be the first vertex on C in topological order
- Let v_j be the vertex just before v_i in C
- v_j is to the right of v_i in topological order $\mathbf{j > i}$
- But $(\mathbf{v_j}, \mathbf{v_i}) \in \mathbf{E}$

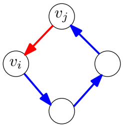


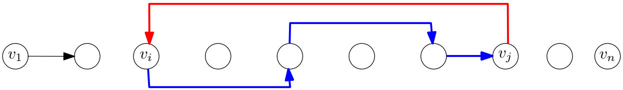
Topological Order

If G has a cycle, then no hope for a topological ordering

> If G has a topological ordering, then G is a DAG

- Suppose G has a topological order and G has a cycle C
- Let v_i be the first vertex on C in topological order
- Let v_j be the vertex just before v_i in C
- v_j is to the right of v_i in topological order $\mathbf{j} > \mathbf{i}$
- But $(\mathbf{v_j}, \mathbf{v_i}) \in \mathbf{E}$





Topological Order

If G is a DAG, then G has a topological ordering

Constructive Proof: Repeatedly add a source in the remaining DAG to list

Algorithm Topological Sorting of a DAG

while $V \neq \emptyset$ **do**
  $v \leftarrow \textsc{FindSource}(V, E)$
  $\textsc{Print}\ v$
  $V \leftarrow V \setminus \{v\}$
  $E \leftarrow E \setminus \{(v, u) \mid u \in V\}$

- In-degrees array can be computed in $O(m)$ time
- FINDSOURCE will take $O(n)$ time (search for 0 degree vertex)
- Total runtime of deleting edges is $O(m)$
- Total runtime is $O(nm)$ ▷ Can be improved a lot

Topological Order

If G is a DAG, then G has a topological ordering

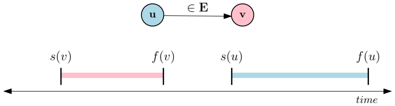
Algorithm Topological Sorting of a DAG

```
IN-DEG[1...n] ← IN-DEGREES(G, V)
v ← INDEX-OF-MIN(IN-DEG)                    ▷ A source must exists if G is DAG
ENQUEUE(Q, v)                                ▷ Q is a queue of sources
while V ≠ ∅ do
    v ← DEQUEUE(Q)
    PRINT v
    for u ∈ N(v) do
        IN-DEG[u] ← IN-DEG[u] − 1
        if IN-DEG[u] = 0 then                ▷ Check if u became a source
            ENQUEUE(Q, u)
    V ← V \ {v}
```

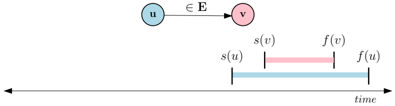
Runtime is $O(n + m)$

Topological Order via DFS

Lemma: If (u, v) is an edge in a DAG, then $f(u) > f(v)$



cross edge



tree or forward edge (check predecessor)

- Check both cases whether DFS first visit u or v
- In either case $f(u) > f(v)$

Corollary: Largest finishing time is for a source

Yields a DFS based algorithm for topological sorting

Topological Order via DFS

write vertices in decreasing order of $f(v)$