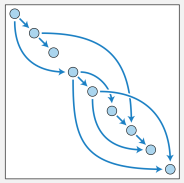
Applications of DFS – Directed Graphs

- **topological sort** – order the vertices of G so that all edges are oriented from an earlier vertex to a later one
  - possible if and only if G is a DAG (directed acyclic graph)
  - algorithm – the ordering is the reverse of the order in which vertex processing is completed (exit time) when dfs is started from a vertex *s* where indeg(*s*) = 0 (i.e. *s* has no incoming edges)

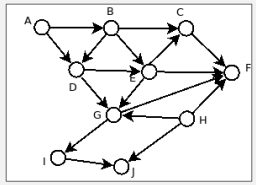


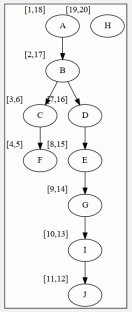
Topological Sort

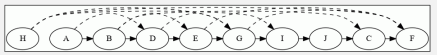
```
time = 0
for each vertex v in V do
  if G.inDegree(v) = 0
    dfs(G,v)

dfs(G,s)
  for each vertex u in V-{s} do
    state[u] = "undiscovered"
    prev[u] = null
  state[s] = "discovered"
  prev[s] = null
  dfshelper(G,s)

dfshelper(G,u)
  time = time+1
  entry[u] = time
  for each edge (u,v) in G.incidentEdges(u) do
    if state[v] = "undiscovered" then
      state[v] = "discovered"
      prev[v] = u
      dfshelper(G,v)
  state[u] = "processed"
  time = time+1
  exit[u] = time
```







Applications of DFS – Directed Graphs

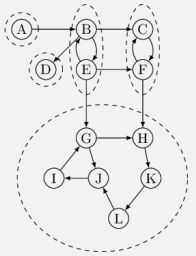
- **topological sort** – order the vertices of G so that all edges are oriented from an earlier vertex to a later one
  - possible if and only if G is a DAG (directed acyclic graph)
  - algorithm – the ordering is the reverse of the order in which vertex processing is completed (exit time) when dfs is started from a vertex *s* where indeg(*s*) = 0 (i.e. *s* has no incoming edges)

---

intuition
- exit timestamp for *u* is after all of the outgoing incident edges (*u*,*v*) have been processed, which means *u*'s exit timestamp is after the exit timestamps of its adacent vertices *v* and *u* occurs before *v* in the topological ordering
- edges are oriented (*u*,*v*) – *u* appears before *v* in the ordering so the edges are correctly oriented

Applications of DFS – Directed Graphs

- **is G strongly connected?** – *strongly connected* means a directed path exists between every pair of vertices
  - algorithm
    - dfs(s), then reverse all of the edges of G and repeat dfs(s) – G is strongly connected if the same set of vertices are discovered/processed each time
- **strongly connected components**
  - an algorithm
    - repeatedly compute the intersection of vertices reachable by dfs(s) and by dfs(s) with the graph's edges reversed, removing each set as a strongly connected component
  - another algorithm
    - repeatedly find a cycle and contract those vertices into a single vertex
    - when there are no more cycles, each remaining vertex represents a different strongly connected component



Takeaways

- DFS algorithm
- DFS-based algorithms / applications
  - graph traversal
  - reachability
  - finding cycles (undirected graphs)
  - cut vertices (undirected graphs)
  - cut edges (undirected graphs)
  - topological sort (directed graphs)
  - strongly connected / strongly connected components (directed graphs)