Directed graphs

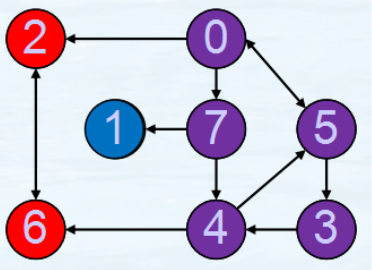
- Connectivity
- Transitive closure: Warshall algorithm
- Topological sort

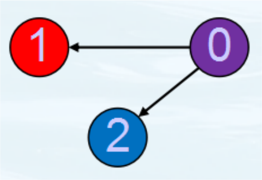
Directed graphs: digraphs

- A digraph is a set of vertices V and a set of directed edges E
  - A directed path is a list of vertices $\{v_i\}$ s.t. $v_i v_{i+1} \in E$
  - t is reachable from s iff there is a directed path from s to t
- A directed acyclic graph (DAG) is a digraph with no directed cycles
  - Vertex with only out-edges: source
  - Vertex with only in-edges: sink
- A digraph is strongly connected iff $\forall u, v \in V$, u is reachable from v
  - A strongly connected component is a maximal strongly connected subgraph

Kernel

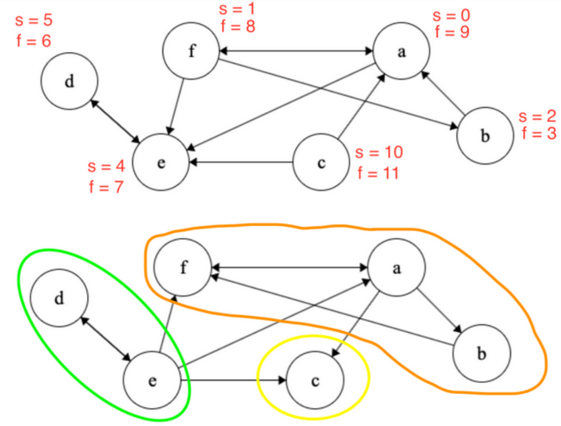
- Given a digraph D, define another digraph $K(D)$
  - each vertex in $K(D)$ maps to a strongly connected component of D
  - $uv \in E(K(D))$ iff there is an edge from the component corresponding to u to v
  - $K(D)$ is called the kernel DAG of D





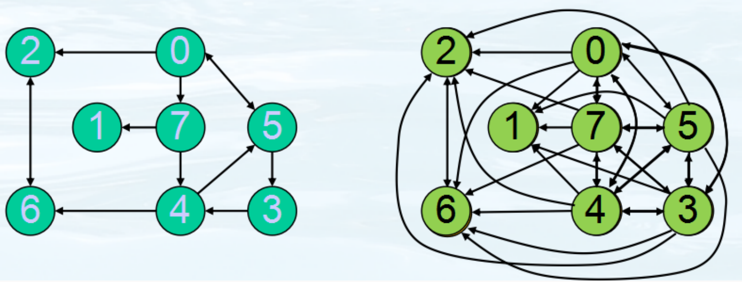
Find strongly connected components (SCC): Kosaraju Algorithm

- Run depth first search on G and keep track of finishing times
- Reverse all the edges in G
- Run depth first search on the reversed graph from the node with the largest finishing time, adding an SCC each time a dead end is reached



Transitive Closure

- Transitive closure of D: a digraph with the same vertices but with an edge from s to t iff t is reachable from s in D



Computing Transitive Closure

- Compute Boolean multiplication of adjacency matrix: A^n
  - Use AND as $\times$, OR as $+$
  - Complexity: $O(n^4)$
- Improvement: compute A^i until converge
- Further improvement: compute $A, A^2, A^4, \cdots$: complexity: $O(n^3 \log n)$

Computing Transitive Closure: Warshall Algorithm

Idea

```
for (every intermediate node i)
  for (every source s)
    for (every destination t)
      if (s reaches i & i reaches t)
        s reaches t;
```

Algorithm

```
for (i = 0; i < n; ++i)
  for (s = 0; s < n; ++s)
    for (t = 0; t < n; ++t)
      if (A[s][i] && A[i][t])
        A[s][t] = 1;
```

Improvement

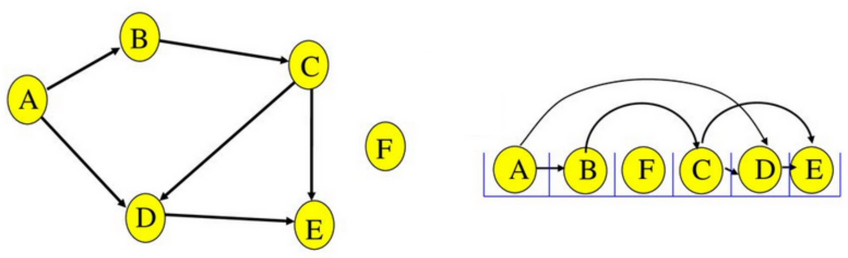
```
for (i = 0; i < V; ++i)
  for (s = 0; s < V; ++s)
    if (A[s][i])
      for (t = 0; t < V; ++t)
        if (A[i][t])
          A[s][t] = 1;
```

Shortest Path: Warshall Algorithm

```
for (i = 0; i < V; ++i)
  for (s = 0; s < V; ++s)
    for (t = 0; t < V; ++t)
      if (A[s][i] + A[i][t] < A[s][t])
        A[s][t] = A[s][i] + A[i][t];
```

Topological Sort

- Given a DAG G, find a total ordering s.t. $\forall uv \in E(G)$, u precedes v in the ordering

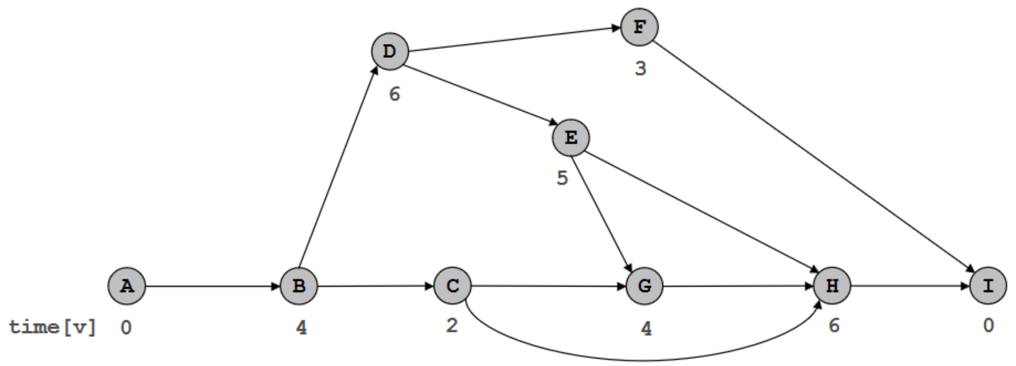


Topological Sort Algorithm

- Step 1: Identify a source s
  - If no source: halt (cycle)
- Step 2: delete s and related edges, enqueue s
- If graph is not empty: goto Step 1
- Complexity: $O(n+m)$

Topological Sort Application: Scheduling

- Task v takes $time[v]$ to execute
- Precedence constraints
- Problem: what is earliest time to finish each task?



Topological Sort Application: Scheduling

- Compute topological order of vertices
- Initialize $fin[v] = 0$ for all v
- Consider v in topological order
  - for each edge $v \to w$, set $fin[w] = \max(fin[w], fin[v] + time[w])$