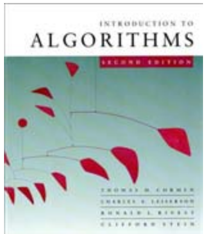
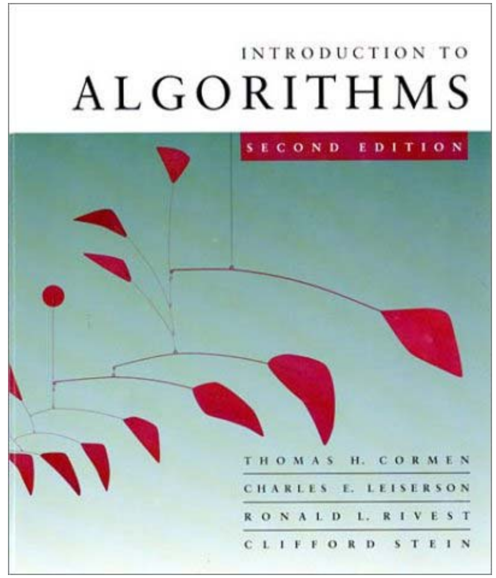
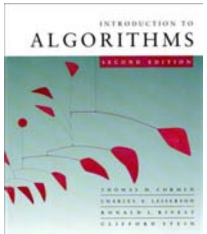


CS 33433 – Fall 2007



Topological Sort

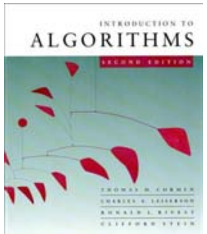
Carola Wenk



Paths, Cycles, Connectivity

Let $G=(V,E)$ be a directed (or undirected) graph

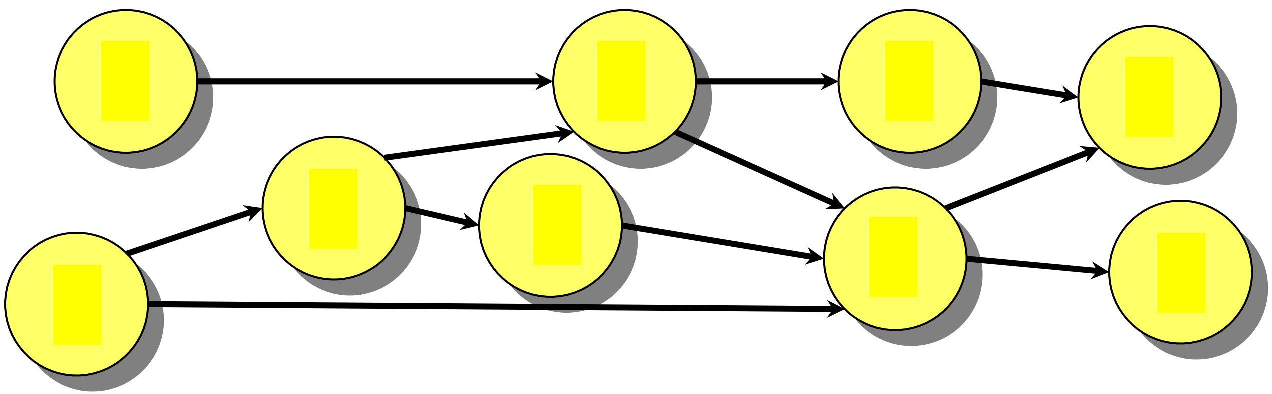
- A **path** from v_1 to v_k in G is a sequence of vertices $v_1, v_2, \ldots, v_k$ such that $(v_i, v_{\{i+1\}}) \in E$ (or $\{v_i, v_{\{i+1\}}\} \in E$ if G is undirected) for all $i \in \{1, \ldots, k\text{-}1\}$.
- A path is **simple** if all vertices in the path are distinct.
- A path $v_1, v_2, \ldots, v_k$ forms a **cycle** if $v_1 = v_k$ and $k \geq 2$.
- A graph with no cycles is **acyclic**.
  - An undirected acyclic graph is called a **tree**. (Trees do not have to have a root vertex specified.)
  - A directed acyclic graph is a **DAG**. (A DAG can have undirected cycles if the direction of the edges is not considered.)
- An undirected graph is **connected** if every pair of vertices is connected by a path. A directed graph is **strongly connected** if for every pair $u, v \in V$ there is a path from u to v and there is a path from v to u.
- The **(strongly) connected components** of a graph are the equivalence classes of vertices under this reachability relation.

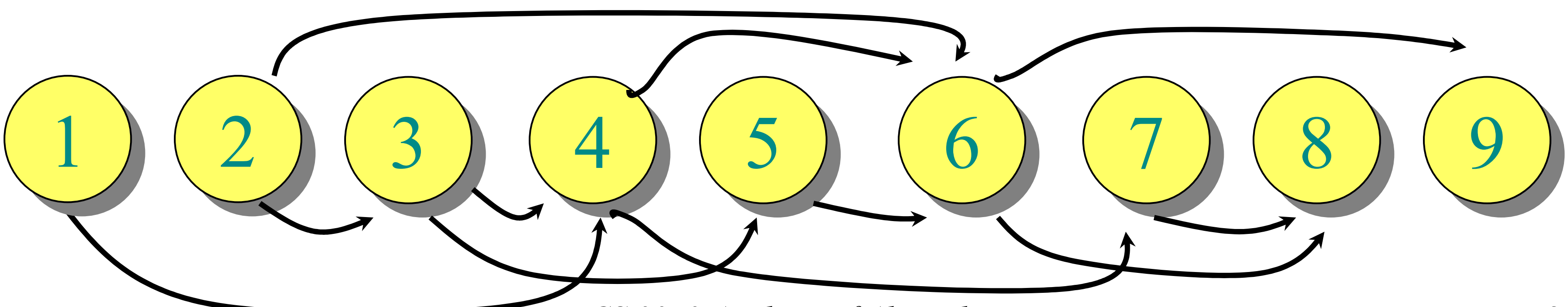


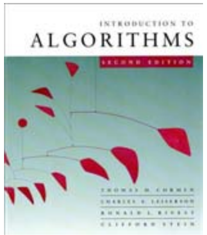
Topological Sort

Topologically sort the vertices of a ***directed acyclic graph*** (***DAG***):

- Determine $f : V \to \{1, 2, \ldots, |V|\}$ such that $(u, v) \in E$ $\Rightarrow f(u) < f(v)$.

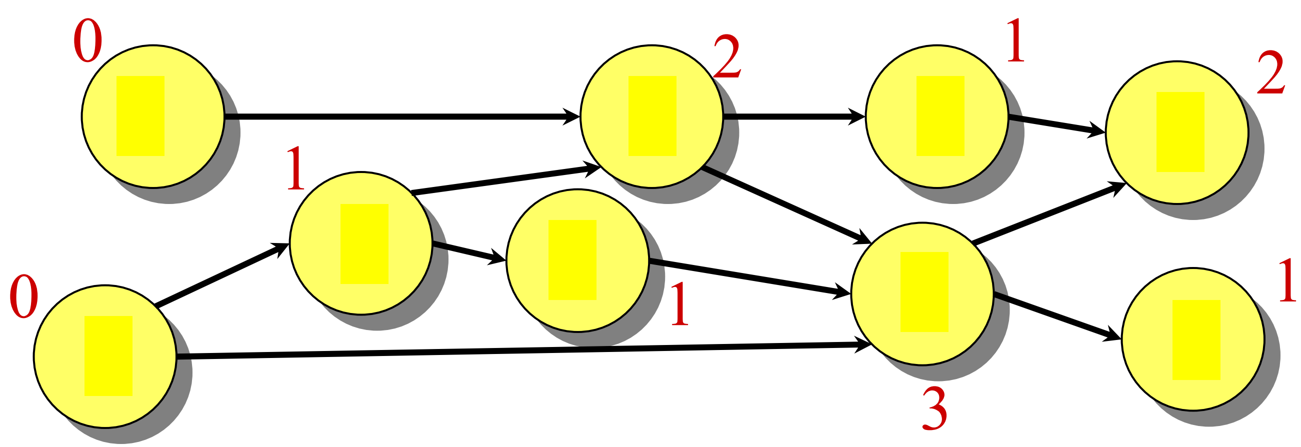


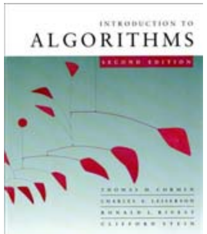




Topological Sort Algorithm

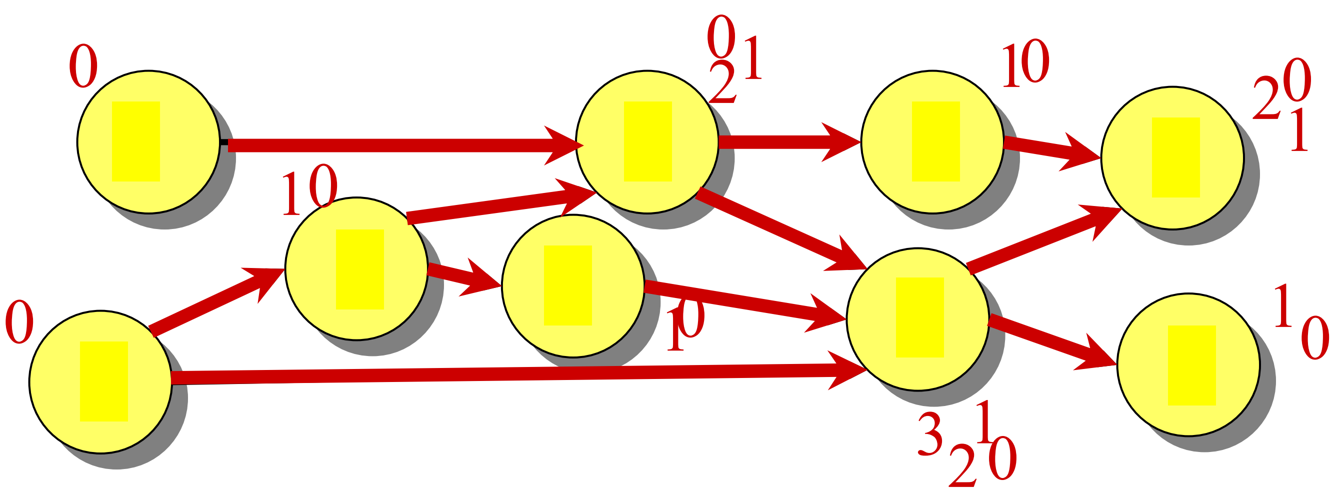
- Store vertices in a priority min-queue, with the in-degree of the vertex as the key
- While queue is not empty
  - Extract minimum vertex v, and give it next number
  - Decrease keys of all adjacent vertices by 1

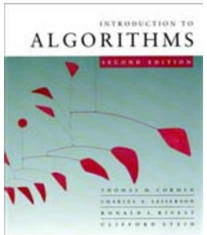




Topological Sort Algorithm

- Store vertices in a priority min-queue, with the in-degree of the vertex as the key
- While queue is not empty
  - Extract minimum vertex v, and give it next number
  - Decrease keys of all adjacent vertices by 1





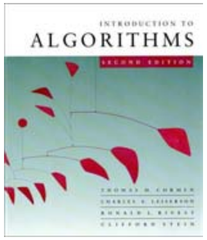
Topological Sort Runtime

Runtime:

- $O(|V|)$ to build heap + $O(|E|)$ DECREASE-KEY ops

$\Rightarrow$ $O(|V| + |E| \log |V|)$ with a binary heap

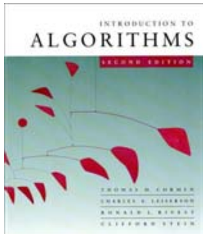
$\Rightarrow$ $O(|V| + |E|)$ with a Fibonacci heap



Depth-First Search revisited

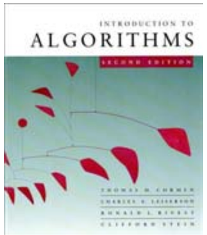
```
DFS(G=(V,E))
    Mark all vertices in G as “unvisited”
    time=0;
    for each vertex v ∈ V do
        if v is unvisited
            DFS_rec(G,v)
```

```
DFS_rec(G, v)
    visit v;
    d[v]=++time; //discover time
    for each w adjacent to v do
        if w is unvisited
            Add edge (v,w) to tree T
            DFS_rec(G,w)
    f[v]=++time; //finish time
```



DFS Edge Classification

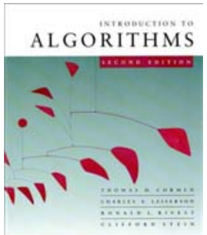
- Edge (u,v) in depth-first forest:
  - **Tree edge:** v was discovered by by exploring edge (u,v)
- Edge (u,v) not in depth-first forest:
  - Back edge: v is ancestor of u in depth-first forest
  - Forward edge: v is descendant of u in depth-first forest
  - Cross edge: Any other edge



DFS-Based Topological Sort Algorithm

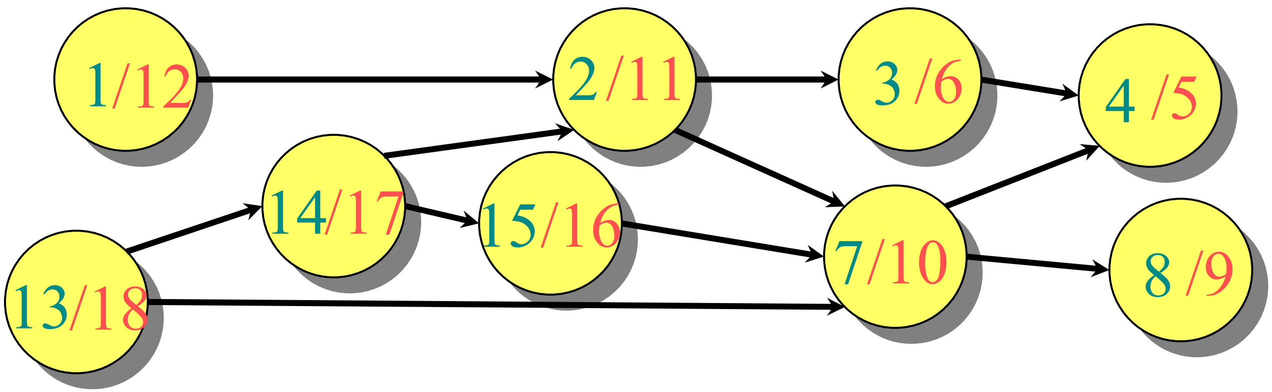
- Call DFS on the directed acyclic graph $G=(V,E)$
  $\Rightarrow$ Finish time for every vertex
- Reverse the finish times (highest finish time becomes the lowest finish time,…)
  $\Rightarrow$ Valid function $f : V \rightarrow \{1, 2, \ldots, |V|\}$ such that
  $(u, v) \in E \Rightarrow f(u) < f(v)$.

Runtime: $O(|V|+|E|)$

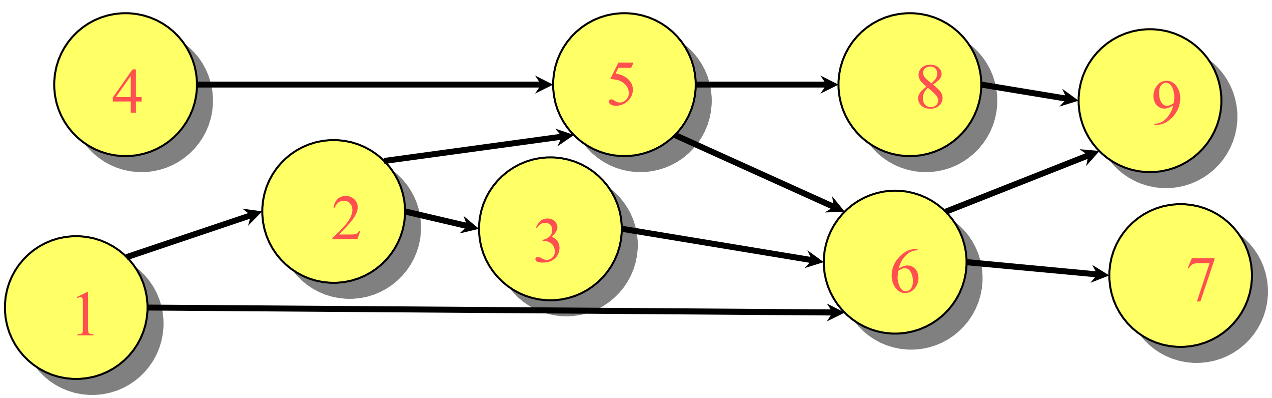


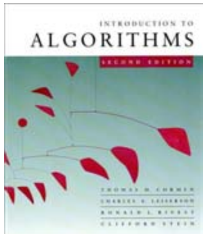
DFS-Based Topological Sort

- Run DFS:



- Reverse finish times:





Topological Sort Runtime

Runtime:

- $O(|V|)$ to build heap + $O(|E|)$ DECREASE-KEY ops

⇒ $O(|V| + |E| \log |V|)$ with a binary heap

⇒ $O(|V| + |E|)$ with a Fibonacci heap

- DFS-based algorithm: $O(|V| + |E|)$