CSE 421: Algorithms

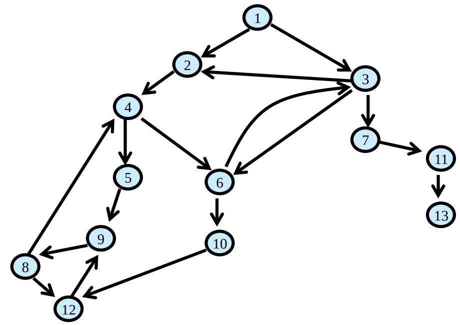
Winter 2014

Lecture 6: Graph traversal and topological sorting

Reading: Sections 3.3-3.6

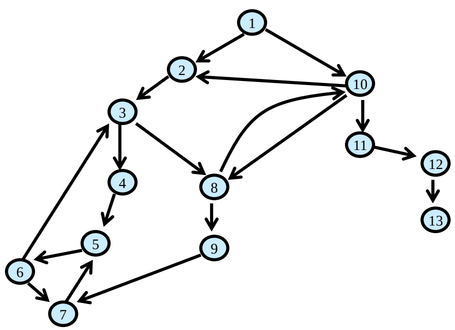
breadth-first search

- Completely explore the vertices in order of their distance from s
- Naturally implemented using a queue



depth-first search

- Completely explore the vertices in DFS order (duh)
- Naturally implemented using recursion



properties of BFS

- BFS(s) visits x if and only if there is a path in G from s to x.
- Edges followed to undiscovered vertices define a "breadth first spanning tree" of G
- Layer i in this tree, L_i
  - those vertices u such that the shortest path in G from the root s is of length i.
- On undirected graphs
  - All non-tree edges join vertices on the same or adjacent layers

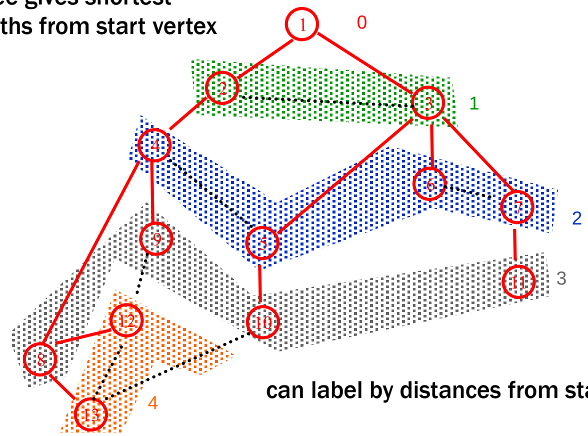
properties of BFS

On undirected graphs:

All non-tree edges join vertices on the same or adjacent layers.

BFS application: shortest paths

Tree gives shortest paths from start vertex



can label by distances from start

connected components

Want to answer questions of the form:

- **Given**: vertices u and v in G
- Is there a path from u to v?

connected components

Want to answer questions of the form:

- **Given**: vertices u and v in G
- Is there a path from u to v?

Idea: create array A such that

$A[u]$ = smallest numbered vertex that is connected to u

- question reduces to whether $A[u] = A[v]$?

connected components

Want to answer questions of the form:

- Given: vertices u and v in G
- Is there a path from u to v?

Q: Why not create an array **Path**[u,v]?

Idea: create array A such that

$A[u]$ = smallest numbered vertex that is connected to u

- question reduces to whether $A[u] = A[v]$?

connected components

- initial state: all **v** unvisited

```
for s ← 1 to n do
    if state(s) ≠ "fully-explored" then
        BFS(s): setting A[u] ←s for each u found
            (and marking u visited/fully-explored)
    endif
endfor
```

- Total cost: $O(n + m)$
  - each vertex is touched once in this outer procedure and the edges examined in the different BFS runs are disjoint
  - works also with depth first search

DFS(u) – recursive version

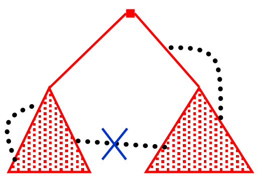
```
Global Initialization: mark all vertices "unvisited"
DFS(u)
  mark u "visited" and add u to R
  for each edge (u,v)
      if (v is "unvisited")
          DFS(v)
  mark u "fully-explored"
```

properties of DFS(s)

- Like BFS(s):
  - DFS(s) visits x if and only if there is a path in G from s to x
  - Edges into undiscovered vertices define a "depth first spanning tree" of G
- Unlike the BFS tree:
  - the DFS spanning tree isn't minimum depth
  - its levels don't reflect min distance from the root
  - non-tree edges never join vertices on the same or adjacent levels
- but…

non-tree edges

- All non-tree edges join a vertex and one of its descendents/ancestors in the DFS tree
- No cross edges.

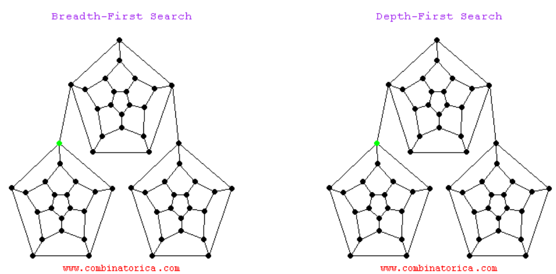


bipartite graph

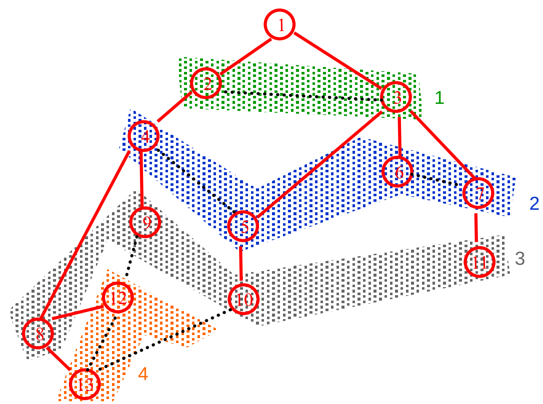
Definition:

Theorem:
Graph is bipartite iff does not contain an odd cycle.

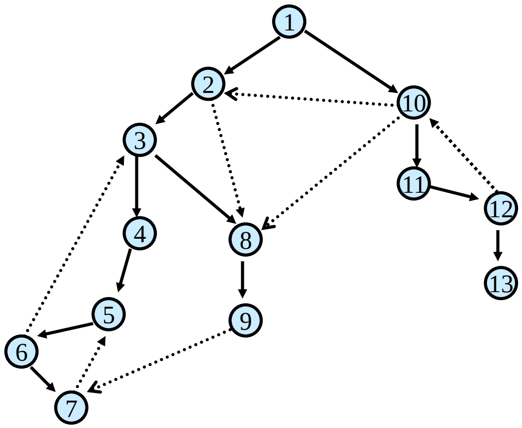
bfs vs dfs



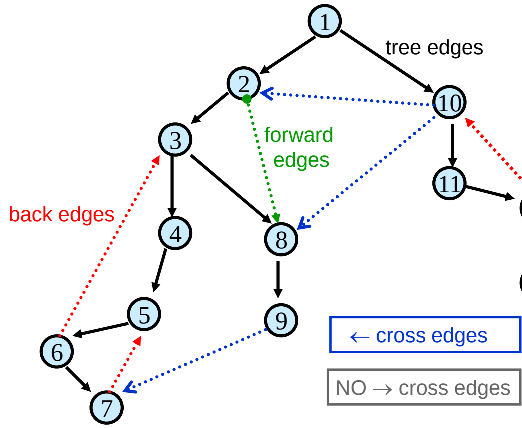
BFS for bipartite testing



DFS(v) for a directed graph



DFS(v)

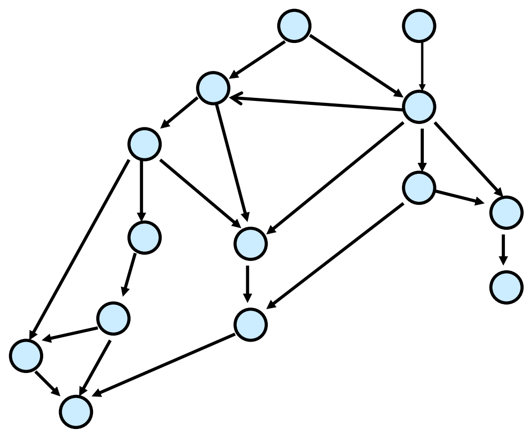


directed acyclic graphs

- A directed graph $G = (V, E)$ is **acyclic** if it has no **directed** cycles

- Terminology: A **d**irected **a**cyclic **g**raph is also called a **DAG**

directed acyclic graph



topological sort

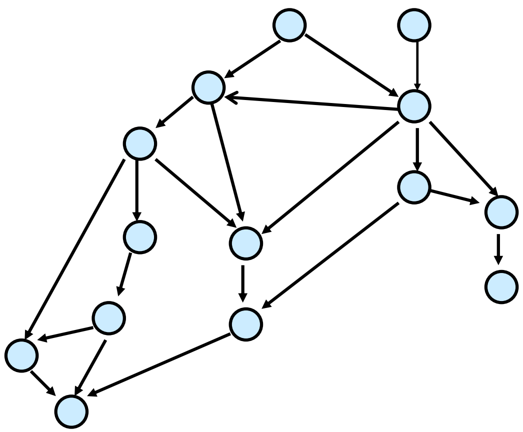
- **Given:** a directed acyclic graph (DAG) **G=(V,E)**
- **Output:** **numbering** of the vertices of **G** with **distinct** numbers from **1** to **n** so edges only go from lower number to higher numbered vertices

- Applications
  - nodes represent tasks
  - edges represent precedence between tasks
  - topological sort gives a sequential schedule for solving them

topological sort

- **Given:** a directed acyclic graph (DAG) **G=(V,E)**
- **Output:** **numbering** of the vertices of **G** with **distinct** numbers from **1** to **n** so edges only go from lower number to higher numbered vertices

directed acyclic graph



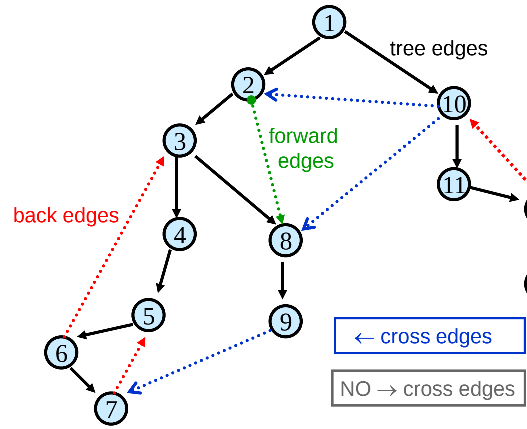
in-degree 0 vertices

Lemma: Every DAG has a vertex of in-degree 0

topological sort

- Can do using DFS

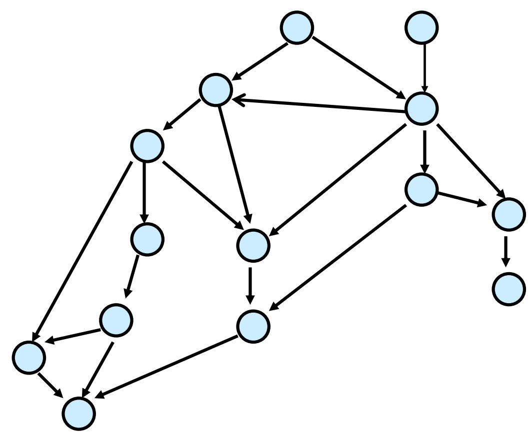
DFS(v)



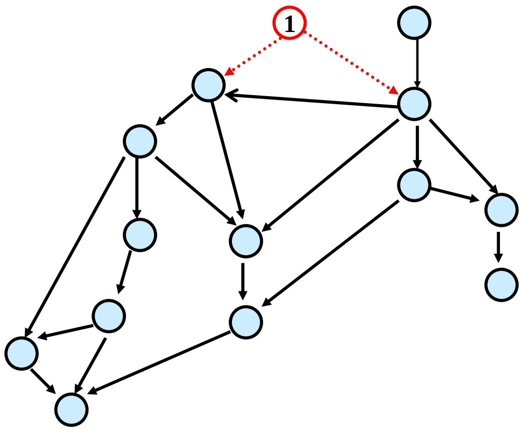
topological sort

- Can do using DFS
- Alternative simpler idea:
  - Any vertex of in-degree 0 can be given number 1 to start
  - Remove it from the graph and then give a vertex of in-degree 0 number 2, etc.

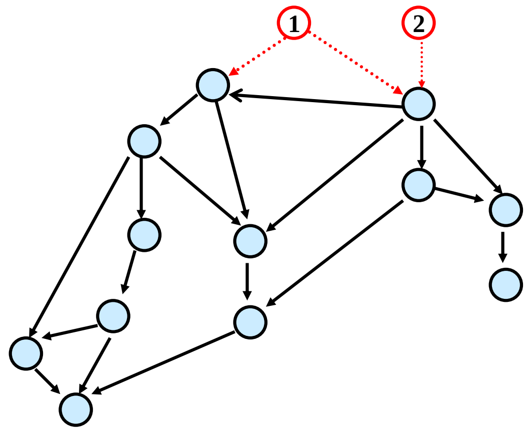
topological sort



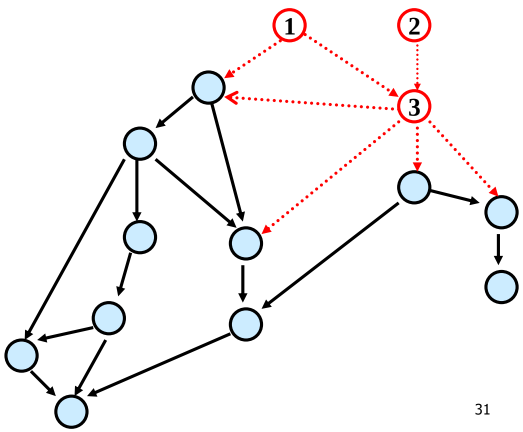
topological sort



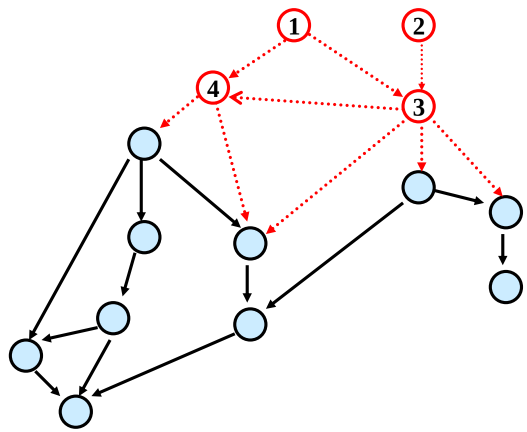
topological sort



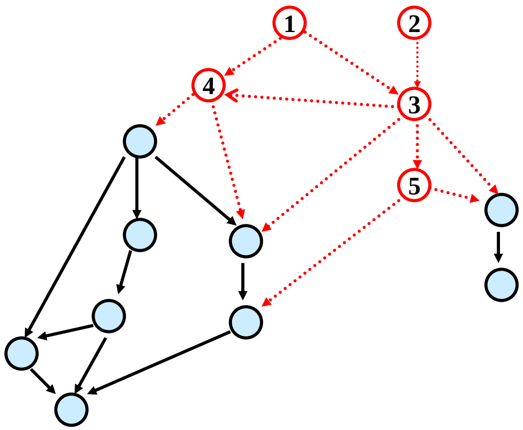
topological sort



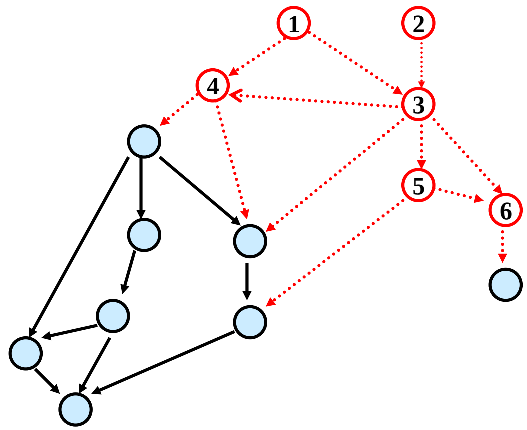
topological sort



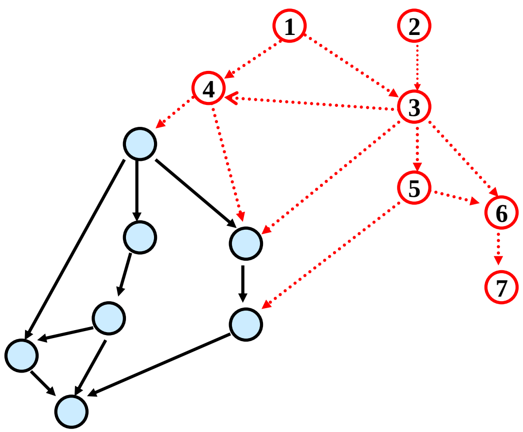
topological sort



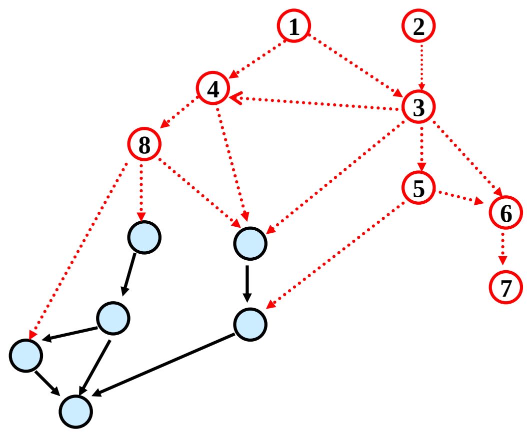
topological sort



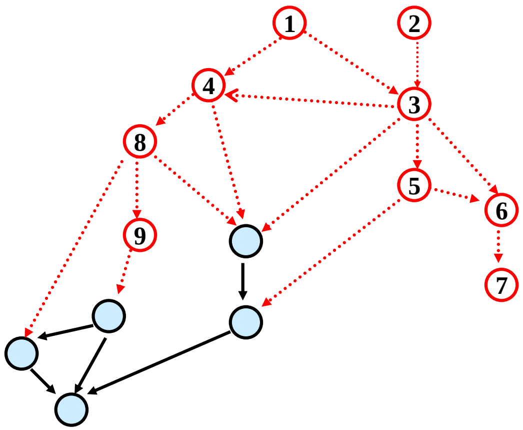
topological sort



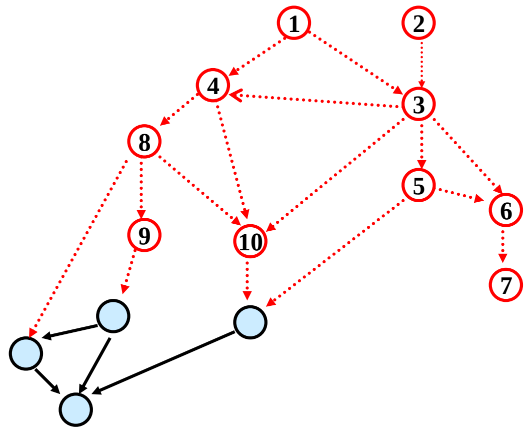
topological sort



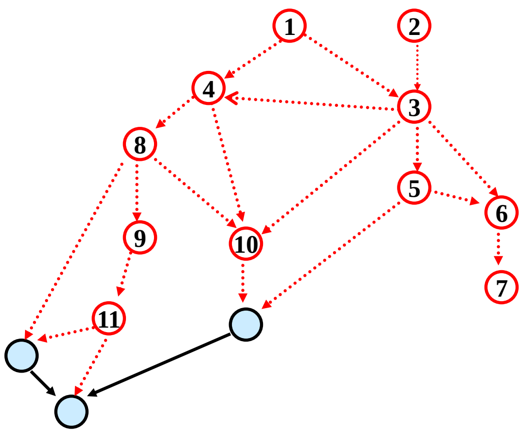
topological sort



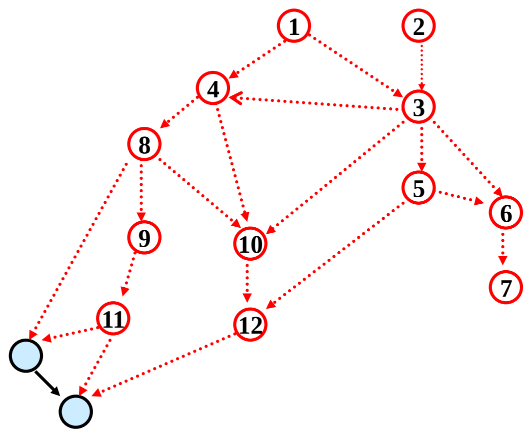
topological sort



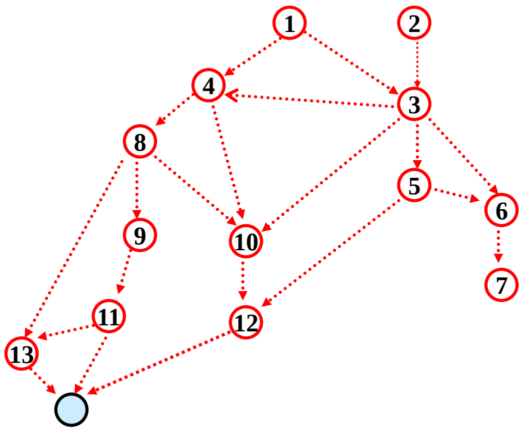
topological sort



topological sort



topological sort



implementing topological sort

- Go through all edges, computing array with in-degree for each vertex $O(m+n)$
- Maintain a queue (or stack) of vertices of in-degree **0**
- Remove any vertex in queue and number it
- When a vertex is removed, decrease in-degree of each of its neighbors by **1** and add them to the queue if their degree drops to **0**

Total cost:

topological sort