Algorithms and Data Structures

Module 1

Lecture 6
Graph traversals: depth-first search, breadth-first search and their applications. Part 3

Adigeev Mikhail Georgievich

mgadigeev@sfedu.ru

adimg@yandex.ru

DFS & BFS: applications

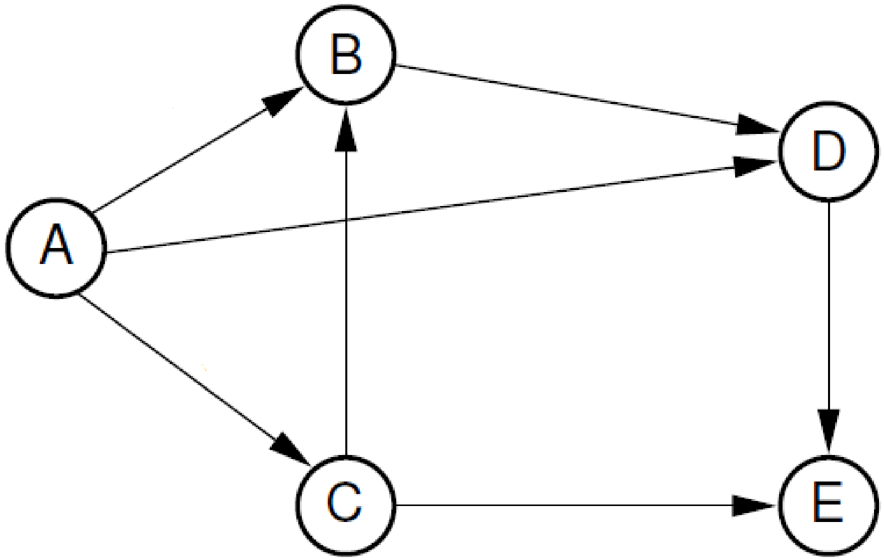
- DFS/BFS:
  - ✓ Connected components detection (see lecture 4)
- BFS:
  - ✓ Calculating distances (see lecture 5)
  - ✓ Bipartiteness testing
- DFS:
  - ✓ Detecting cycles
  - ✓ Topological ordering (topological sort) of a DAG

BFS: Calculating distances

Graph G=(V,E).

A *distance* between vertices *u* and *v* is the minimum length of the path between *u* and *v*.

dist(A,E) = 2



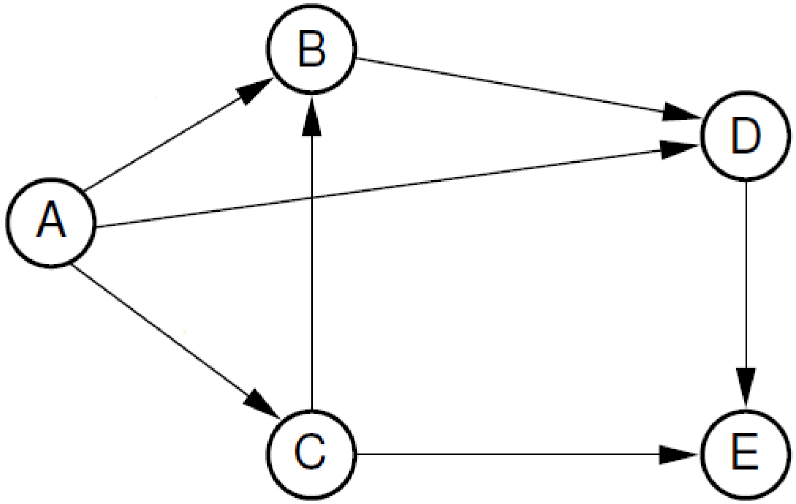
BFS: Calculating distances

Problem: for given $G(V,E)$ and a vertex $s \in V$ find distances and the shortest paths from s to every other vertex.

```
DistancesBFS(G)
// Initialization
Create d[],p[]
For each v ∈ V\{s}:
        d[u] = +∞;
        p[u]= null;
d[s] = 0;
Enqueue(s)
```

BFS: Calculating distances

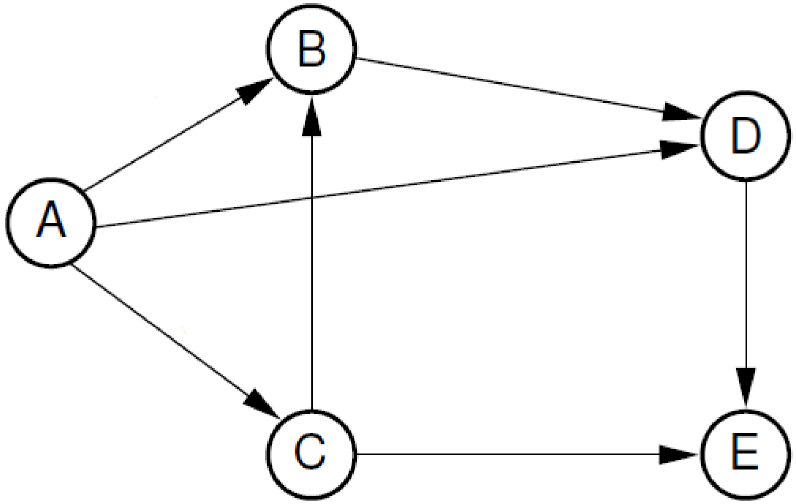
```
// Breadth-First Search
While (Queue is not empty):
      v = Dequeue()
      if v is unvisited:
            Mark v as 'visited'
            For each u in Adj(v):
                  if d[u] > d[v]+1:
                        d[u] = d[v]+1
                        p[u] = v
                        Enqueue(u)
```



BFS: Calculating distances

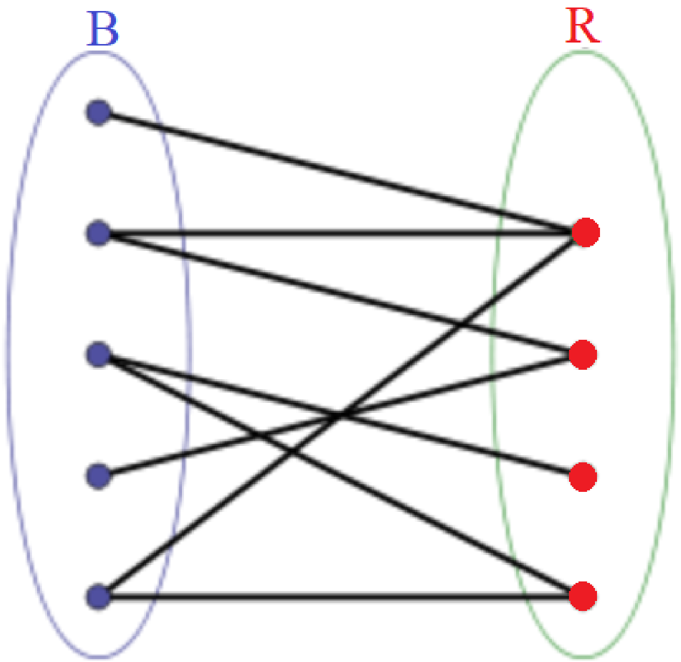
How do we construct a path from s to v?

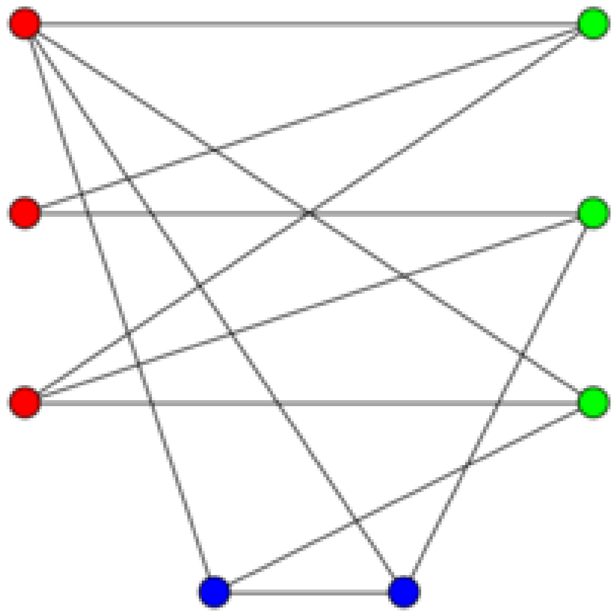
We start from v and reconstruct the path backward to s: we move from a current vertex u to $x = p[u]$, then to $y = p[x]$,… , until we get s.



BFS: Bipartiteness check

Graph $G(V, E)$ is called ***bipartite*** iff its vertex set V can be partitioned into two disjoint subsets (***parts***): $V = B \cup R$ such that for each edge $e \in E$ the endpoints of e belong to different subsets.

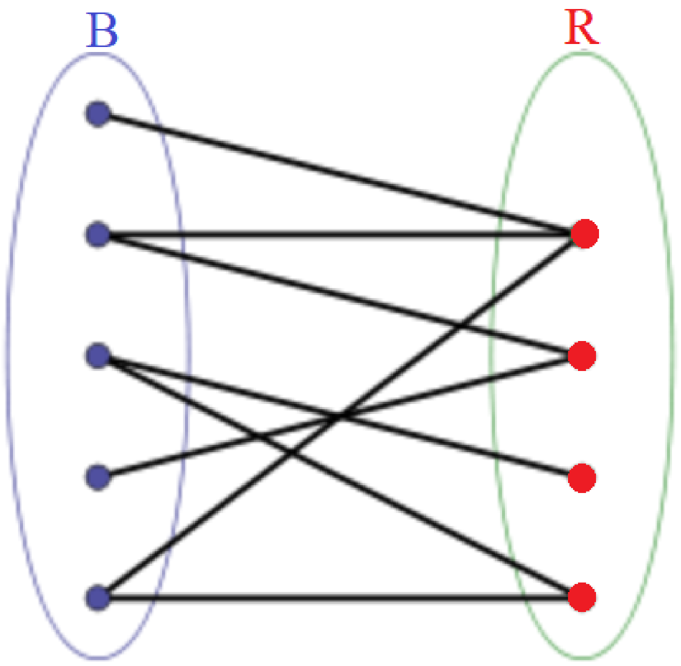


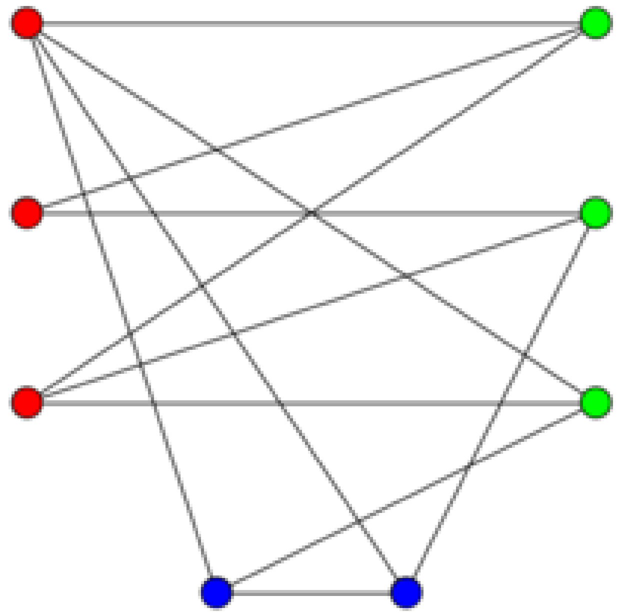


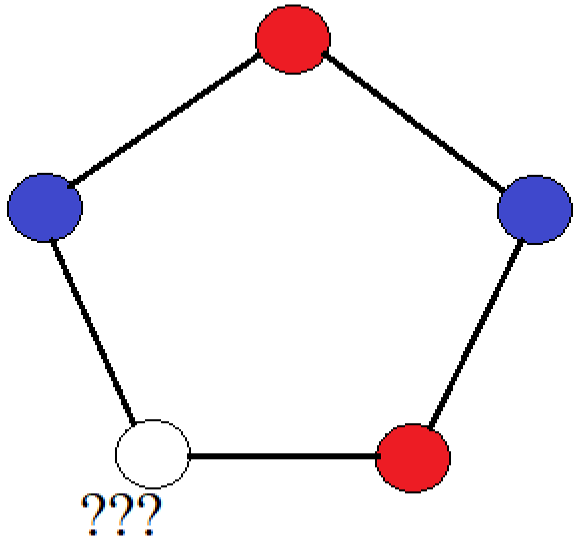
BFS: Bipartiteness check

Theorem. Graph $G(V,E)$ is ***bipartite*** iff it has no cycles of odd length.

Corollary: trees and forests are bipartite graphs.







BFS: Bipartiteness check

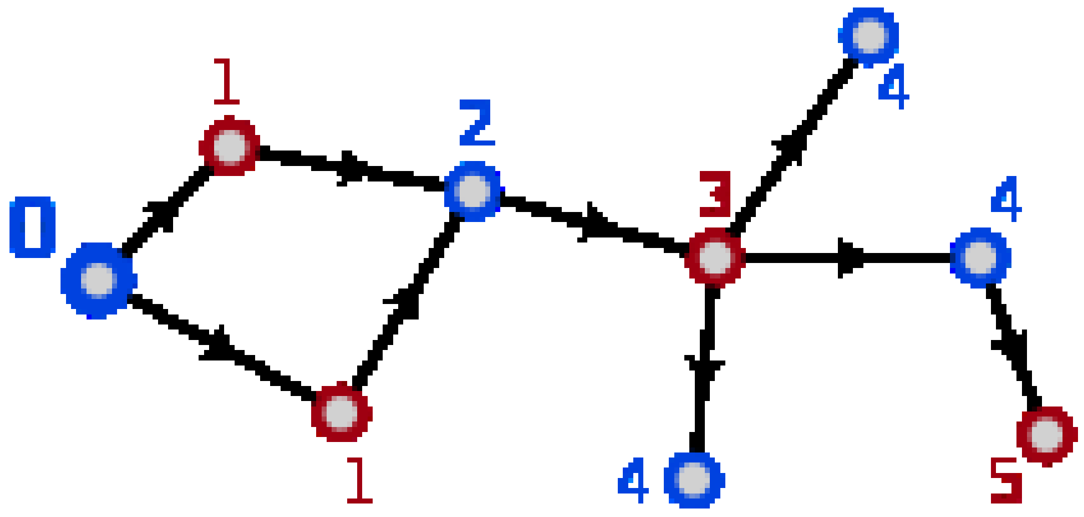
Algorithm for bipartiteness check.

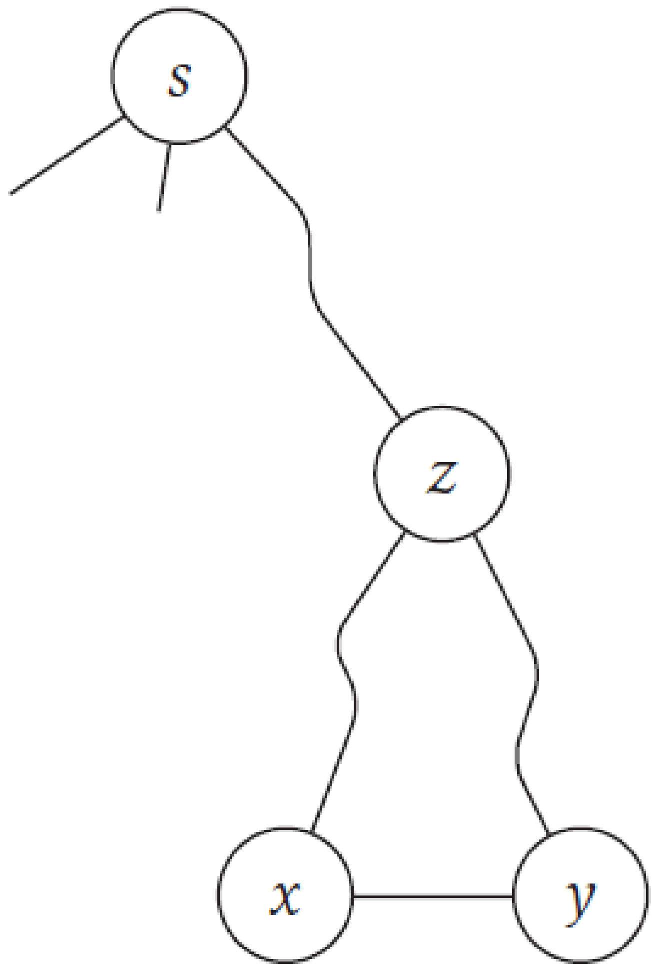
Let $G(V, E)$ be a connected graph.

1. $R = B = \emptyset$
2. Select any $s \in V$. d[s]=0.
3. Calculate $d[v]$ - distances from s to all other vertices.
4. For each $v \in V$:
   - if d[v] is odd: $R = R \cup \{v\}$
   - else: $B = B \cup \{v\}$
5. Scan thru E and check whether the condition holds.

Time complexity: $O(|V| + |\mathrm{E}|)$

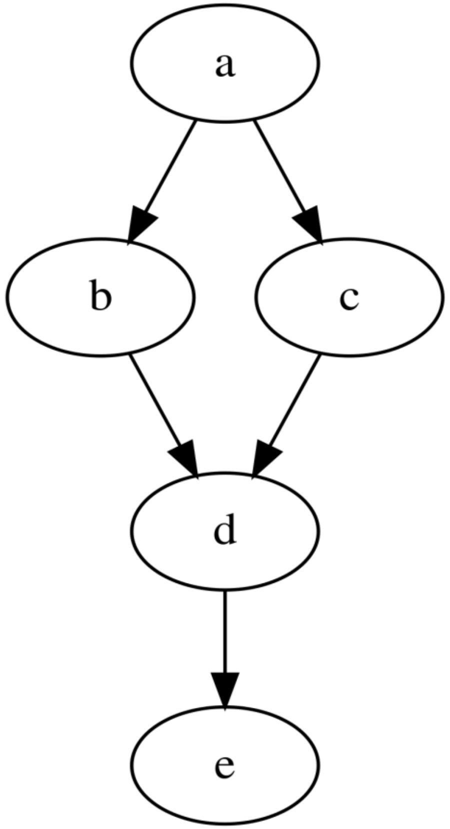
BFS: Bipartiteness check

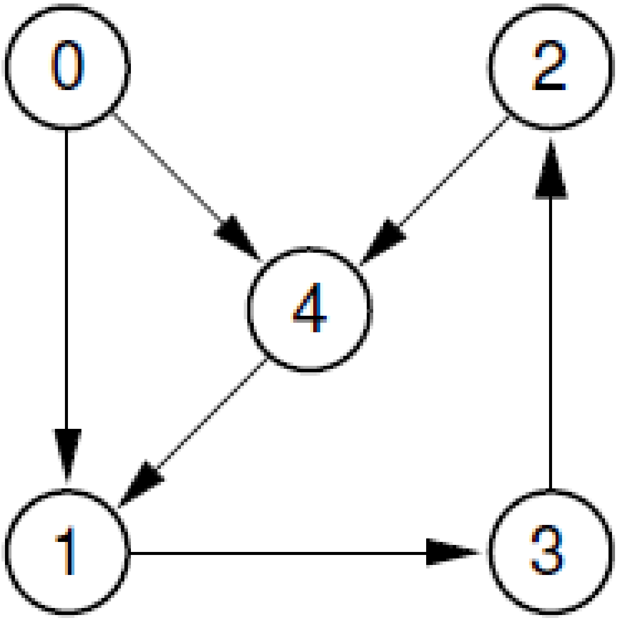




DFS: Detecting cycles

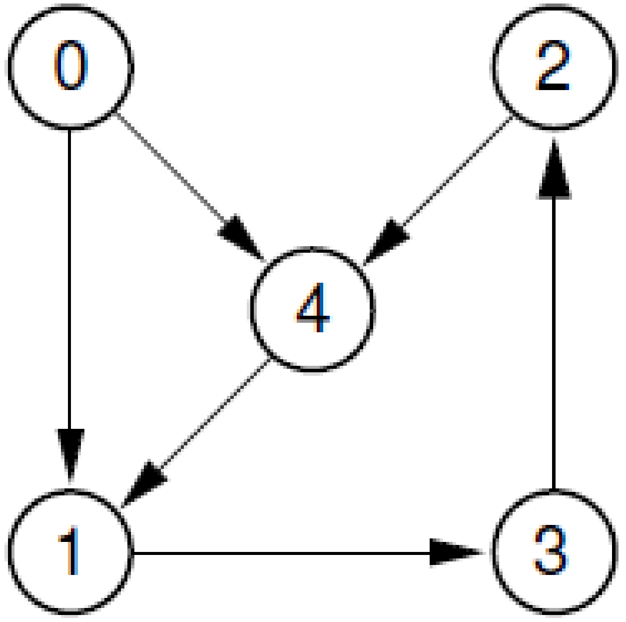
DAG = directed acyclic graph = directed graph with no directed cycles.





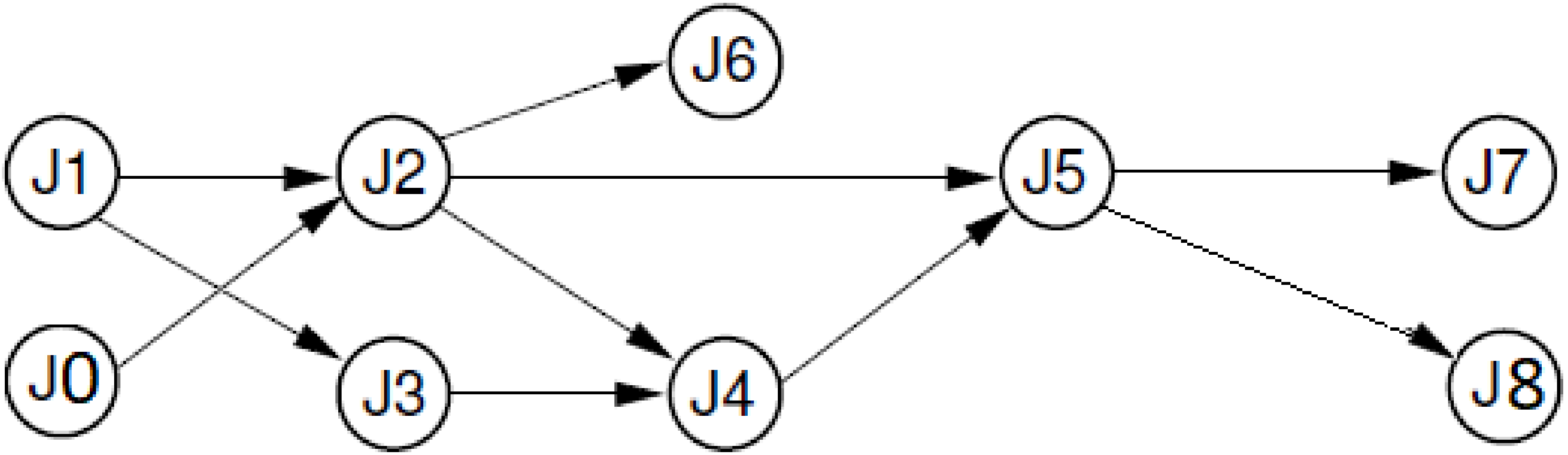
DFS: Detecting cycles

```
DFS(v)
Mark v as 'visited'
Mark v as 'active'
For each u in Adj(v):
      if u is unvisited:
            DFS(u)
      else if u is 'active':
            a cycle found!!!
Mark v as 'inactive'
```



DFS: Topological sort of a DAG

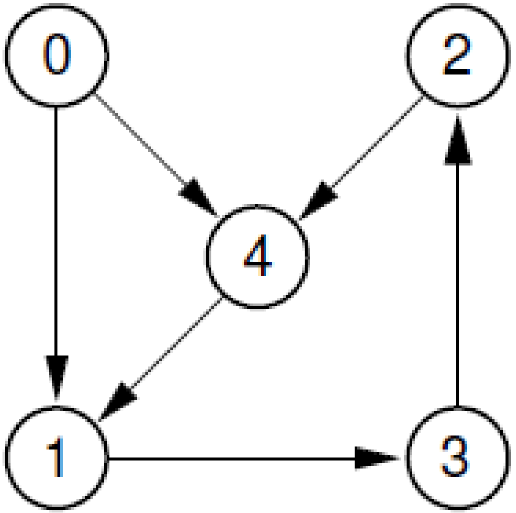
Topological ordering (sort) is vertex numbering $\tau: V \leftrightarrow \{1, \ldots, |V|\}$:
there are no edges (u,v) in G: $\tau(u) > \tau(v)$.



Graphs: definition (lecture 03)

$v \in V$:

- ✓ $\deg(v)$ – ***degree*** of vertex v = number of edges incident to v .
- ✓ $\text{outdeg}(v)$– out-degree of vertex v = number of edges which start from v .
- ✓ $\text{indeg}(v)$– in-degree of vertex v = number of edges which end at v .
- ✓ v is a ***source*** iff $\text{indeg}(v) = 0$
- ✓ v is a ***sink*** iff $\text{outdeg}(v) = 0$



DFS: Topological sort of a DAG

Assign a vertex 'topological number' just before leaving this vertex: initialize `CurTopNum` with $n = |V|$, then run DFS:

```
DFS(v)
~~PreVisit(v)~~
Mark v as 'visited'
For each u in Adj(v):
    if u is unvisited: DFS(u)
PostVisit(v)
```

```
PostVisit(v)
TopNum[v] = CurTopNum
CurTopNum--
```

DFS: Topological sort of a DAG