Outline: Search II: DFS (II of 2)
- depth-first search
- edge classification
- cycle testing
- topological sort

Recall:
- graph search: explore a graph
e.g. find a path from start vertex $s$
to a desired vertex
- adjacency lists: array $\text{Adj}$ of $|V| \text{ linked lists}$
  - for each vertex $u \in V$, $\text{Adj}[u]$ stores $u$'s neighbors, i.e. $\{v \in V \mid (u, v) \in E\}$
  - just outgoing edges if directed

![Graph example]

- $\text{BFS}$: explore level-by-level from $s$
- find shortest paths
Depth-first search (DFS): like exploring a maze
- follow path until you get stuck
- backtrack along breadcrumbs until reach unexplored neighbor
- recursively explore
- careful not to repeat a vertex

parent = \{s: None\}

\[\text{DFS-visit}(s, \text{Adj}):\]
for \(v\) in \(\text{Adj}[s]\):
if \(v\) not in parent:
  parent[\(v\)] = \(s\)
  \(\text{DFS-visit}(v, \text{Adj})\)

\[\text{DFS}(V, \text{Adj}):\]
parent = \{\}
for \(s\) in \(V\):
if \(s\) not in parent:
  parent[\(s\)] = None
  \(\text{DFS-visit}(s, \text{Adj})\)
Example:

Edge classification:
- tree edges (formed by parent)
- nontree edges

- back edge: to ancestor
- forward edge: to descendant
- cross edge (to another subtree)

- to compute this classification, mark nodes for duration they are “on the stack”
- only tree & back edges in undir. graph

Analysis:
- DFS-visit gets called with a vertex $s$ only once (because then parent[$s$] set)
  $\Rightarrow$ time in DFS-visit $= \sum_{s \in V} |\text{Adj}[s]| = O(E)$
- DFS outer loop adds just $O(V)$
  $\Rightarrow O(V + E)$ time (linear time)
Cycle detection: graph $G$ has a cycle \iff DFS has a back edge

**Proof:**

$(\Leftarrow)$

- Tree edges is a cycle
- Back edge: to tree ancestor

$(\Rightarrow)$

Consider first visit to cycle:

- Before visit to $v_i$ finishes, will visit $v_{i+1}$ (\& finish):
  - Will consider edge $(v_i,v_{i+1})$
  - \Rightarrow visit $v_{i+1}$ now or already did
- Before visit to $v_0$ finishes, will visit $v_k$ (\& didn't before)
- Before visit to $v_k$ (or $v_0$) finishes, will see $(v_k,v_0)$ as back edge.
Job scheduling: given directed acyclic graph (DAG), where vertices represent tasks & edges represent dependencies, order tasks without violating dependencies.

Source = vertex with no incoming edges = schedulable at beginning (A, G, I)

Attempt: BFS from each source: slow... and wrong!
- from A finds A, BH, C, F
- from D finds D, BE, CF
- from G finds G, H
- from I finds I

Topological sort: reverse of DFS finishing times (time at which DFS-Visit(v) finishes)

\[
\begin{align*}
\text{DFS-Visit}(v) & \quad \text{order. append}(v) \\
& \quad \text{order. reverse}(c)
\end{align*}
\]
Correctness: for any edge \((u,v)\),
- \(u\) ordered before \(v\)
  - i.e. \(v\) finished before \(u\)

\[
\begin{array}{c}
\circ \rightarrow \bullet \\
\end{array}
\]

- if \(u\) visited before \(v\):
  - before visit to \(u\) finishes,
    will visit \(v\) (via \((u,v)\) or otherwise)
  \implies v \text{ finishes before } u

- if \(v\) visited before \(u\):
  - graph is acyclic
  \implies u \text{ can't be reached from } v
  \implies \text{ visit to } v \text{ finishes before visiting } u
\]
\]