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 $|V|$ 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
- e.g. $\text{Adj}$

- 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

\[ \text{parent}^+ = \{s: \text{None}\} \]

\[
\text{DFS-visit}(s, \text{Adj})::
\]

\[
\text{for } v \text{ in Adj}[s]:
\]

\[
\text{if } v \text{ not in parent:}
\]

\[
\text{parent}^+[v] = s
\]

\[
\text{DFS-visit}(v, \text{Adj})
\]

\[
\text{DFS}(V, \text{Adj}):
\]

\[
\text{parent}^+ = \{\}
\]

\[
\text{for } s \text{ in } V:
\]

\[
\text{if } s \text{ not in parent:}
\]

\[
\text{parent}^+[s] = \text{None}
\]

\[
\text{DFS-visit}(s, \text{Adj})
\]

search from start vertex \( s \)
(only see stuff reachable from \( s \))

explore entire graph
(could do same to extend BFS)
Example:

```
        S1
       /   \
  a     b  c
 /     /   |
v     v   v
 d     e  f
```

*forward edge*  *back edge*  *cross edge*

**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:
- 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)

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

\[
\begin{array}{c}
\text{If } u \text{ visited before } v: \\
\text{before visit to } u \text{ finishes, will visit } v \text{ (via } (u,v) \text{ or otherwise)}
\end{array}
\]
\(\Rightarrow \) \(v\) finishes before \(u\)

\[
\begin{array}{c}
\text{If } v \text{ visited before } u: \\
\text{graph is acyclic}
\end{array}
\]
\(\Rightarrow \) \(u\) can't be reached from \(v\)
\(\Rightarrow \) visit to \(v\) finishes before visiting \(u\)
\]