Depth-First Search

- Review breadth-first search (bFS)
- Present depth-first search (dFS)
- Present topological sort algorithm

Next Time: Garbage Collection

Tuesday: Review
BFS(V source)

For each u ∈ V
    u.discovered = False
    u.dist = ∞
    u.parent Edge = null
Source.discovered = true
Source.dist = 0
queue.enqueue(Source)

While (!queue.isEmpty())
    u = queue.dequeue()
    For each edge e in outgoing edges from u
        v = e.dest
        If (!v.discovered)
            v.discovered = true
            v.dist = u.dist + 1
            v.parent Edge = e
            queue.enqueue(v)
Called breadth-first search because of how search proceeds.

In general, short wide tree.
depth-first search

1. Use stack vs queue

2. Continue (re-starting if needed) until all vertices are visited

3. Keep “time” when each vertex discovered + finish processing all outgoing edges

4. Replace boolean discovered by a 3-valued variable, color
false for discovered

- color white - not yet discovered
  - discovery time (not on stack yet)
  - color gray - on stack but not
    - finishing time - done processing
  - color black - done processing (finished)
top-level DFS (in a graph class)

def dfs():
    color all vertices white and set parent to null
    time = 1
    for each vertex u in graph
        if (u.color == white)
            dfsVisit(u)

    u is root

    each call here will create one tree in dfs forest
\text{dfsVisit}(\text{Vertex } u) \\
\text{U. color} = \text{gray} \\
\text{U. discovery Time} = \text{time} + t \\
\text{for all vertices } v \text{ reachable from } u \text{ by an edge} \\
\text{if } v\text{.color} = \text{white} \\
\text{U. parent} = u \\
\text{dfsVisit}(v) \\
\text{if } v\text{.color} = \text{gray} \\
\text{what does this tell you? you Found a cycle} \\
\text{U. color} = \text{black} \\
\text{U. finishing Time} = \text{time} + t \\
\text{(can be constructed using parent edges)}
Time complexity
\[ O(m+n) \] with adjacency list

call DFS visit exactly once for each vertex,
dominant cost is iterating over outgoing edges
Task Scheduling

Precedence Graph for Changing a Flat Tire

- Drive off
- Put lug tool away
- Tighten lugs
- Put car lower
- Put lug on

- Put jack away
- Remove jack

- Put old tire in trunk
- Put spare on
- Remove spare from trunk

- Get lug tool
- Get owner's manual

- Remove lugs
- Remove old tire
- Jack up car

- Take out jack
- Place jack under car
Topological Sort

Find an ordering of vertices in a directed acyclic graph so that if edge $u \rightarrow v$ then $u$ precedes $v$ in ordering. Should report a cycle if the graph has no valid order.
Observation: Valid topological order is defined by the reverse order of finishing times.
DFS visit call(W)

If edge $u \rightarrow v$ we are looking at and $v$ is gray, there is a cycle.

Stack top $u$
If there is a directed cycle, then at some point an edge will be encountered that goes gray vertex \( U \rightarrow V \) grey cycle which first has discovered vertex.
If graph is acyclic, reverse order of finishing times is a valid topological order.

Stack before \( u \) finishes: \( W, U \).

Suppose I've drawn in a valid order.
Strongly Connected Components.

Sets of vertices as big as possible where each vertex in the set is reachable from all others.