Admin: 

Rivest

Announce: Talk by me: "Security of Voting Systems", 7pm, 32-123 10/16/08

Reading: CLRS 22.1-22.3, B.4

Outline:  
- intro - graph searching
  - Breadth-first search (BFS)
  - Depth-first search (DFS)

Graph searching:

Given: finite graph \((G = (V, E))\) & start vertex \(s \in V\)  
(assume directed, although need not be)

Explore: visit every vertex reachable from \(s\).

1. \(s\) is reachable from \(s\).
2. if \(u\) is reachable from \(s\), and \(v \in \text{Adj}[u]\) then \(v\) is reachable from \(s\)  
   (go to \(u\), then follow edge \((u, v)\))
3. only vertices reachable from \(s\) are those named, so by 1 

if \(u\) is reachable from \(s\), then
there is a path
\[ s \rightarrow v_1 \rightarrow v_2 \rightarrow \cdots \rightarrow v_k \rightarrow u \]  
(of length \(k+1\)) leading from \(s\) to \(u\). We'll find such paths...
Exploring a graph

Let \( R = \text{set of vertices known to be reachable from start vertex } s \)

Let \( Q = \text{vertices known to be reachable from } s \),
which we haven't yet visited (applied (2))
\( Q \subseteq R \) always (\( Q = \text{"frontier"} \))

\[
\begin{align*}
R &= \text{set([s])} \\
Q &= \text{set([s])} \quad \text{or } Q = \text{queue([s])}
\end{align*}
\]

while \( Q \):
\[
\begin{align*}
\text{u} &= Q.\text{dequeue()} \\
\text{for } v \text{ in Adj}[u]: \\
&\quad \text{if } v \text{ not in } R: \\
&\qquad R, \text{add}(v) \\
&\qquad Q, \text{add}(v)
\end{align*}
\]

now \( R \) is set of vertices reachable from \( s \)
\( Q \) could be FIFO, LIFO, other...

Running time:
- each vertex added to \( R \) and \( Q \) at most once
- each adjacency list examined at most once
- \( \sum_{u} |\text{Adj}[u]| = |E| \) by definition
- running time is \( \Theta(V+E) \) \underline{linear time}
We can keep track of paths, too:

- if $v$ is discovered from $u$,
  then call $u$ the "parent" of $v$.
- Keep track of parents:

  \[
  p[s] = \text{nil} \quad s \text{ has no parent} \\
  p[v] = u \quad u \text{ is parent of } v \quad u \rightarrow v
  \]

**Example:**

\[
\begin{array}{ccccccccccc}
S & a & s & d & c & f & w \\
\end{array}
\]

\[
\begin{array}{ccccccccccc}
R = \{s, a, x, z, d, c, f, w\} \\
Q = [s, a, x, z, d, c, f, w] \quad (d \text{ cross out}) \\
\end{array}
\]

\[
\rightarrow \text{ edges form a tree, with paths leading to all} \\
\text{reachable vertices} \\
\]

(Above search is actually BFS...)
BFS (Breadth-First Search)

- So far, we have been talking about generic search. It finds all reachable vertices (by induction) 10/10/08 but we can't say much more...
- By ensuring that Q is FIFO ("first-in first-out"), we obtain BFS.
- BFS enables us to calculate shortest path distance from s to each reachable vertex.
  - Let $d[v]$ be this distance.
  - Initially, $d[s] = 0$
  - When we add $v$ to $R$, set $d[v] = d[v] + 1$
- (Show on example of page 3)
- Why does this work?
  - We visit all vertices at distance $k$ before any at distance $k+1$
  - When we visit all vertices at distance $k$, we discover all vertices at distance $k+1$
  - (Each vertex at distance $k+1$ is reachable from some vertex at distance $k$)
  - Can think of BFS as exploring one layer after another
    - Layer $k$ = all vertices at distance $k$ from $s$
      - "Breadth first" = all of layer $k$, before any of layer $k+1$
- BFS finds shortest path from $s$ to each reachable vertex.
  (Good for Pocket Cube!)
- (We'll generalize BFS later in course, when edges have weights (i.e. lengths).)
- Note: book uses color at each vertex: white: unseen gray: in Q black: $R=Q$
DFS (Depth-First Search)

- Searches deeper before returning to explore earlier alternatives.
- For undirected graph, corresponds to feasible "physical search" (exploring a maze). With BFS, you need to "teleport" to get to each vertex removed from Q. With DFS, you just retrace your steps (backtrack).
- Can implement DFS by changing Q from FIFO to LIFO, although we won't explore that insight here...

DFS example:
- each square a vertex
- ↑ explore ↓ backtrack

- follow path until you get stuck
- backtrack until you reach an unexplored edge;
  - explore it (recursively)
- careful not to repeat a vertex
• Code for DFS is very simple

```python
def visit(u):
    for v in Adj[u]:
        if v not in R:
            R.add(v)
            visit(v)

R = set()
for u in V:
    if u not in R:
        R.add(u)
        visit(u)
```

• 2nd version guarantees exploring entire graph...
• Can keep track of each node's parent, as we did for BFS.
  Got tree of edges leading from s (root) to all vertices
  reachable from s. Or, a set of trees containing all vertices
• Running time is $O(V+E)$ [each vertex visited at most once]
• Example on directed graph,

![Directed Graph Diagram]

- Forward edge
- Back edge
- Cross edge

• 2 trees generated: one rooted at a, one at c
• Next time: applications of DFS