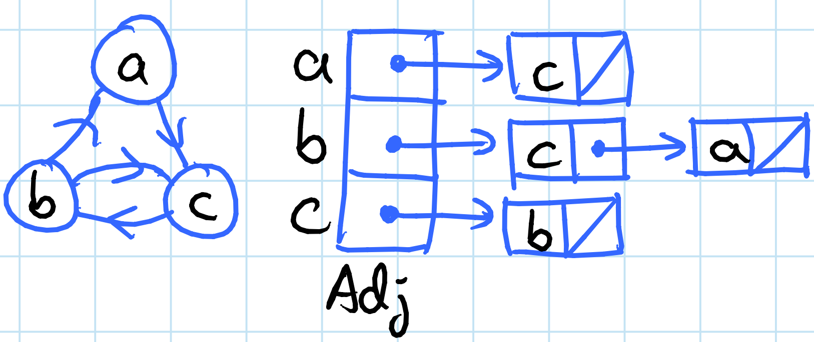
6.006	Lecture 13	Mar. 20, 2008

Outline: Search II
- breadth-first search
- shortest paths
- depth-first search
- edge classification

Reading: CLRS 22.2-22.3

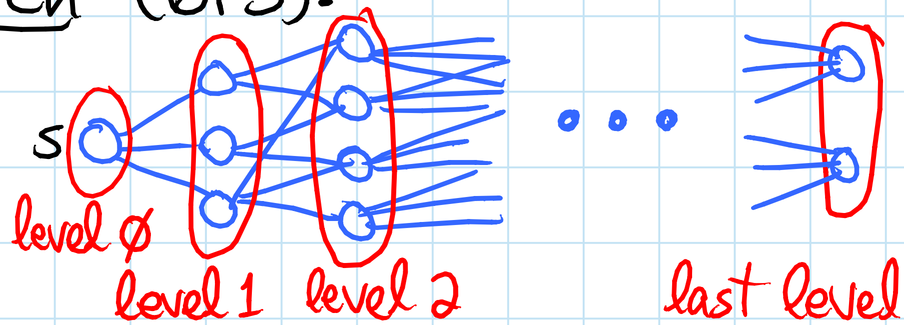
Recall:
- graph search: explore a graph
  e.g. find a path from start vertex s to a desired vertex
- adjacency lists: array Adj of $|V|$ linked lists
  - for each vertex $u \in V$, Adj[u] stores u's neighbors, i.e. $\{v \in V \mid (u,v) \in E\}$
    just outgoing edges if directed

e.g.



Breadth-first search (BFS):

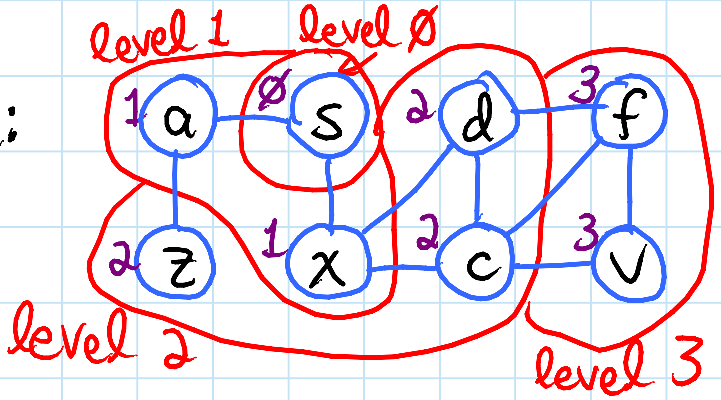
explore graph
level by level
from s



- level ∅ = {s}
- level i = vertices reachable by path of i edges but not fewer
- build level i>∅ from level i−1 by trying all outgoing edges, but ignoring vertices from previous levels

```
BFS(V, Adj, s):
    level = {s: ∅}
    parent = {s: None}
    i = 1
    frontier = [s]            # previous level, i-1
    while frontier:
        next = []             # next level, i
        for u in frontier:
            for v in Adj[u]:
                if v not in level:     # not yet seen
                    level[v] = i       # = level[u]+1
                    parent[v] = u
                    next.append(v)
        frontier = next
        i += 1
```

Example:



$frontier_\emptyset = \{s\}$
$frontier_1 = \{a, x\}$
$frontier_2 = \{z, d, c\}$
$frontier_3 = \{f, v\}$
(not x, c, d)

Analysis:

- vertex v enters next (& then frontier) only once (because level[v] then set)
  - base case: $v = s$

$\Rightarrow$ Adj[v] looped through only once

- time $= \sum_{v \in V} |Adj[v]| = \begin{cases} |E| & \text{for directed graphs} \\ 2|E| & \text{for undirected graphs} \end{cases}$

$\Rightarrow O(E)$ time

- $O(V+E)$ to also list vertices unreachable from v (those still not assigned level)

"LINEAR TIME"

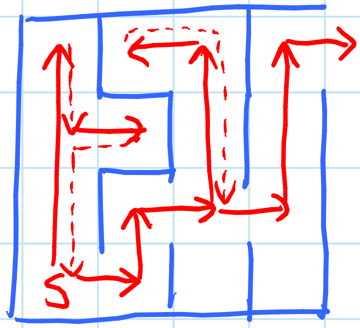
Shortest paths:

- for every vertex v, fewest edges to get from s to v is $\begin{cases} \text{level}[v] & \text{if } v \text{ assigned level} \\ \infty & \text{else (no path)} \end{cases}$
- parent pointers form shortest-path tree = union of such a shortest path for each v

$\Rightarrow$ to find shortest path, take v, parent[v], parent[parent[v]], etc., until s (or None)

Depth-first search (DFS): like exploring a maze

- follow path until you get stuck
- backtrack along breadcrumbs until reach unexplored neighbor
- recursively explore




```
parent = {s: None}

DFS-visit(V, Adj, s):
    for v in Adj[s]:
        if v not in parent:
            parent[v] = s
            DFS-visit(V, Adj, v)
```

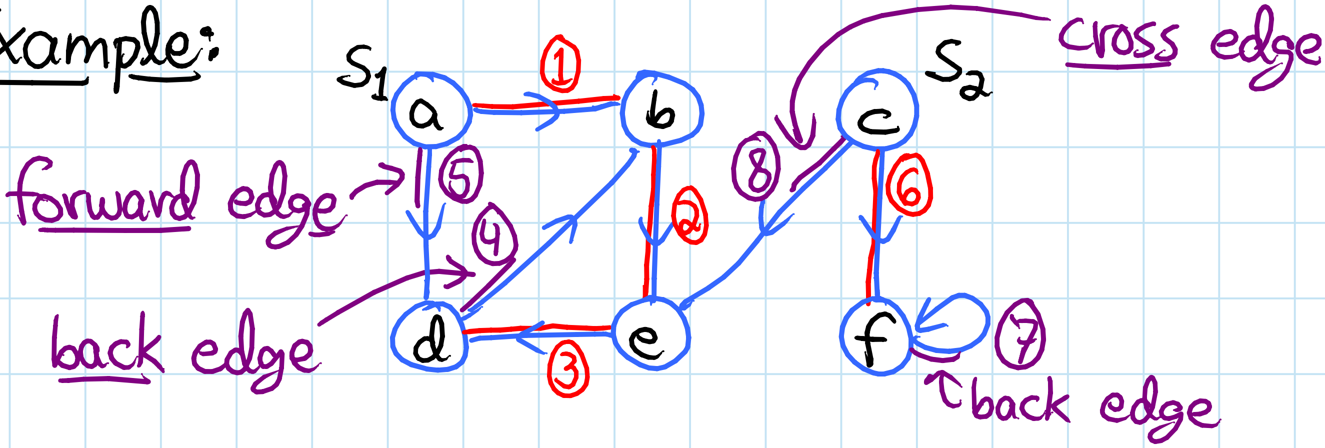
search from start vertex s (only see stuff reachable from s)

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

explore entire graph

(could do same to extend BFS)

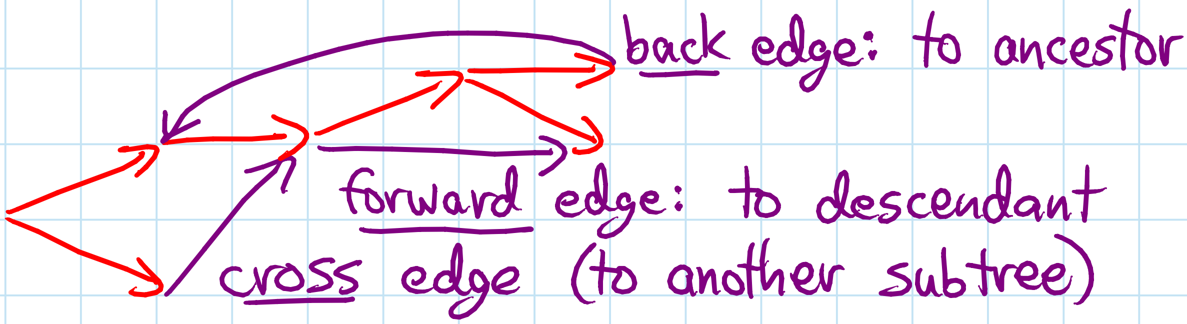
Example:



Edge classification:

tree edges (formed by parent)

nontree edges



- to compute this classification, keep global time counter & store time interval during which each vertex is on recursion stack

Analysis:

- DFS-visit gets called with a vertex s only once (because then parent[s] set)

$\Rightarrow$ time in DFS-visit $= \sum_{s \in V} |Adj[s]| = O(E)$

- DFS outer loop adds just $O(V)$

$\Rightarrow O(V+E)$ time (linear time)