6.006 Introduction to Algorithms
Spring 2008

Lecture 13: Searching II: Breadth-First Search and Depth-First Search

Lecture Overview: Search 2 of 3

- Breadth-First Search
- Shortest Paths
- Depth-First Search
- Edge Classification

Readings

CLRS 22.2-22.3

Recall:

graph search: explore a graph
e.g., find a path from start vertices to a desired vertex

adjacency lists: array Adj of $| V |$ linked lists

- for each vertex $u \epsilon V$, Adj$[u]$ stores u's neighbors, i.e. $\{v \epsilon V \mid (u, v) \epsilon E\}$
  v - just outgoing edges if directed

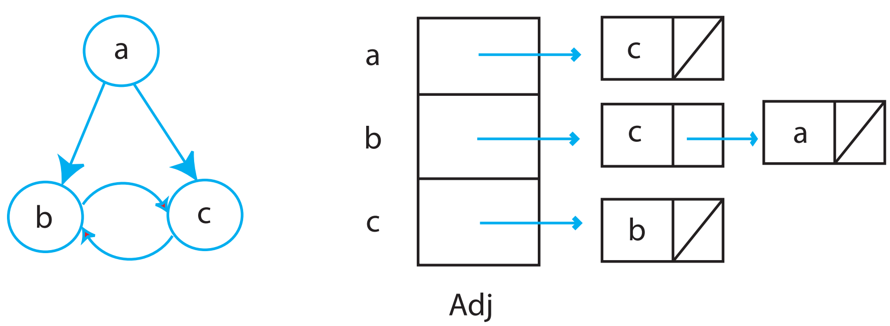


Figure 1: Adjacency Lists

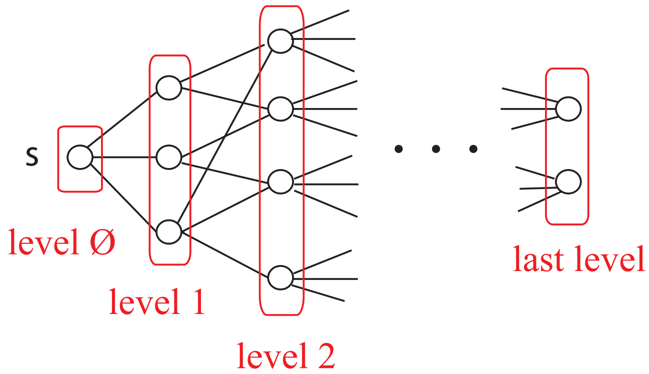


Figure 2: Breadth-First Search

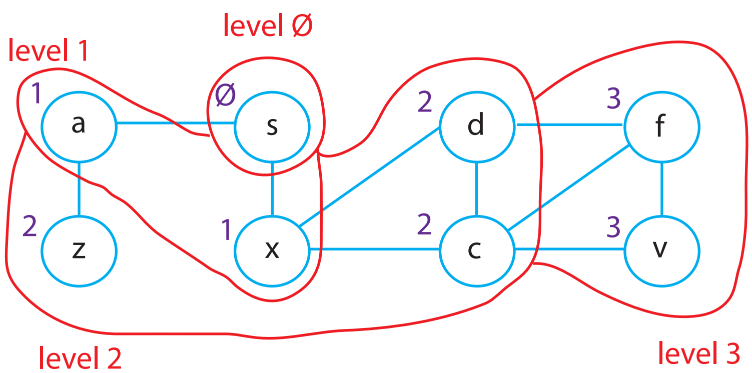
Breadth-first Search (BFS):

See Figure 2
Explore graph level by level from S

- level $\phi = \{s\}$
- level i = vertices reachable by path of i edges but not fewer
- build level $i > 0$ from level $i - 1$ by trying all outgoing edges, but ignoring vertices from previous levels

```
BFS (V,Adj,s):
    level = { s: φ }
    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$_Ø$ = {s}
frontier$_1$ = {a, x}
frontier$_2$ = {z, d, c}
frontier$_3$ = {f, v}
(not x, c, d)

Figure 3: Breadth-First Search Frontier

Analysis:

- vertex V enters next (& then frontier)
  only once (because level[v] then set)

  base case: $v = s$

- $\implies$ Adj[v] looped through only once

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

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

Depth-First Search (DFS):

This is like exploring a maze.

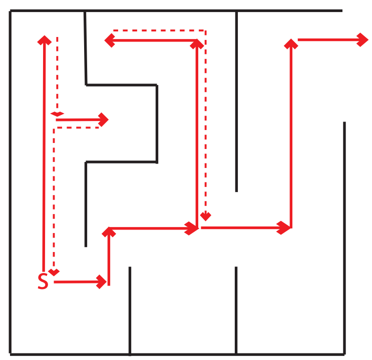


Figure 4: Depth-First Search Frontier

- 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)

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

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

explore entire graph

(could do same to extend BFS)

Figure 5: Depth-First Search Algorithm

Example:

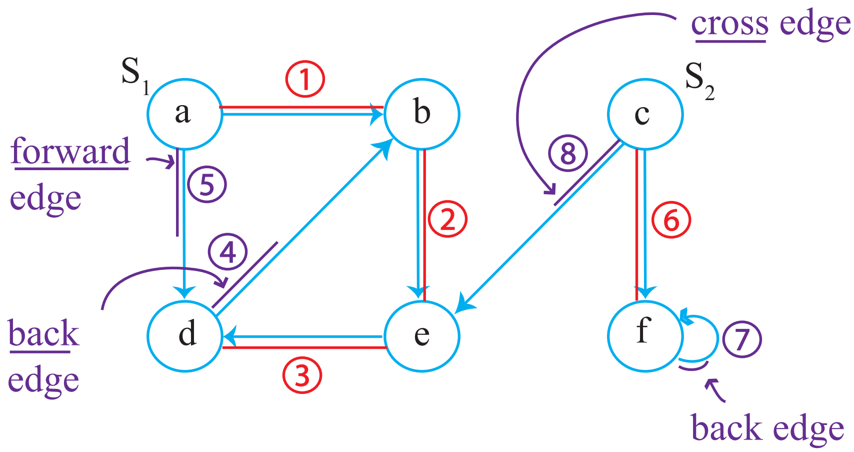


Figure 6: Depth-First Traversal

Edge Classification:

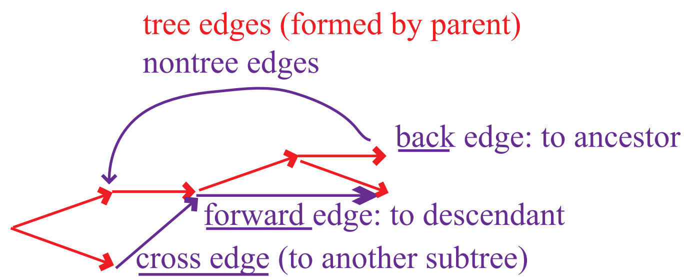


Figure 7: Edge Classification

To compute this classification, keep global time counter and 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)
  $\Longrightarrow$ time in DFS-visit $= \sum_{s \epsilon V} |\ \mathrm{Adj}[s]\ | = O(E)$
- DFS outer loop adds just $O(V)$
  $\Longrightarrow$ $O(V+E)$ time (linear time)