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∈V, Adj[u] stores u’s neighbors, i.e. \{v∈V | (u,v)∈E\}
    v - just outgoing edges if directed

![Adjacency Lists Diagram]

Figure 1: Adjacency Lists
Breadth-first Search (BFS):

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: $\phi$ }
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:

```plaintext
Lecture 13 Searching II 6.006 Spring 2008

Example:

dfgsdfg
vcxz
1 Ø 2 3
322 1
level Ø level 1
level 2 level 3
frontierØ  = {s}
frontier1  = {a, x}
frontier2  = {z, d, c}
frontier3  = {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

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

- \( O(E) \) 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.

- 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)
```

Figure 4: Depth-First Search Frontier

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:

![Diagram](image)

**Figure 6: Depth-First Traversal**

**Edge Classification:**

- tree edges (formed by parent)
- nontree edges
- back edge: to ancestor
- forward edge: to descendant
- cross edge (to another subtree)

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

- DFS outer loop adds just $O(V)$
  \[ \Rightarrow O(V + E) \text{ time (linear time)} \]