1 Why Depth-First Search?

Previously, we saw BFS, which measured distance of each vertex from some starting point. The “opposite” of BFS is DFS.

- DFS visits every node in $G$
- Purpose is to mark each vertex in $G$ with a “visit time” – creates a depth-first ordering of vertices $G$.
- Ordering useful for, e.g., topological sort
- Can also detect and mark cycles in $G$

2 Pseudocode

- Given directed graph $G = (V, E)$, execute DFS starting from every vertex in $V$. (In some sequential order, not in parallel.)
- Each vertex $v$ has a “start time” $s[v]$ and a “finish time” $f[v]$ (both $> 0$)
- Time is incremented globally whenever we start or finish working on a vertex.
- vertex states
  - undiscovered: $s[v] = 0$
  - in-progress: $s[v] > 0, f[v] = 0$
  - finished: $f[v] > 0$
- vertices processed in LIFO (stack) order; will implement via recursion
- Top-level procedure forces all vertices in $G$ to be visited, even if not connected.

```
DFS(G)
   for $u \in V$ do
      $s[u] \leftarrow 0$
      $f[u] \leftarrow 0$
      parent[$u$] $\leftarrow$ null
```
time ← 1
for $u \in V$ do
    if $s[u] = 0$
        $\triangleright$ undiscovered
        DFSVisit($G, u$)
end

- Recursive DFSVisit takes care of an entire connected component.

DFSVisit($G, u$)
    $s[u] \leftarrow$ time
    time++
    $\triangleright$ start $u$
    for $v \in Adj[u]$ do
        if $s[v] = 0$
            $\triangleright$ $v$ not visited yet
            parent[$v$] $\leftarrow u$
            DFSVisit($G, v$)
            $\triangleright$ recur before continuing adj list
        end
    end
    $f[u] \leftarrow$ time
    time++
    $\triangleright$ finish $u$
3 Example

Here’s a quick example of DFS so you can see how it works.

Notice that we explore as far as possible from each vertex, rather than going one step at a time as in BFS.

- **Cost**: \( \Theta(n) \) to initialize
- DFSVisit is called once per vertex (when first discovered): \( \Theta(n) \)
- As with BFS, every edge out of each vertex is checked once (during its processing): \( \Theta(m) \)
- Total cost: \( \Theta(n + m) \)

4 What the Heck is the Point?

We’ll look at a couple of DFS applications.

- Given a directed graph \( G \), how can you tell if \( G \) has a cycle?
- “Looking” at \( G \) is not enough – not automated!
- Cycle could be as long as \( n - 1 \) edges
- Fortunately, DFS has built-in cycle detection!

**Thm**: a digraph \( G \) is cyclic iff DFSVisit finds an in-progress node (start > 0, finish = 0) in its for loop.

- **First**, argue that if in-progress node found, cycle exists.
- Suppose that, while expanding \( u \), we find some in-progress vertex \( v \in \text{Adj}[u] \)
  - Obviously, edge \((u, v)\) exists.
  - Claim there must also be a path from \( v \) to \( u \). Why?
  - Current search path must start from \( v \) (since \( v \) is in-progress), and it has reached \( u \).
- **Second**, argue that if cycle exists, in-progress node will be found
• Some vertex \( v \) in cycle is discovered first (at lowest time).

• Subsequent search from \( v \) will visit every other vertex in cycle for first time before \( v \) is finished (all reachable from \( v \), none seen yet)

• Let \( u \) be predecessor of \( v \) in cycle; in particular, \( u \) will be discovered before \( v \) is finished.

• Hence, traversing edge \((u, v)\) will find \( v \) while it is still in progress. QED

5 Topological Sort

An extension of cycle detection does something useful even when there’s no cycle.

• If \( G \) is cyclic, report it.

• Otherwise (\( G \) is a DAG), find an ordering for the vertices in \( G \) s.t. if \((u, v) \in E\), then \( u \) is ordered before \( v \).

• (Ordering may not be unique!)

Here’s the algorithm:

1. Run DFS on \( G \)
2. If DFS finds a cycle, report “cyclic”
3. Else, output vertices of \( G \) in order from largest to smallest finishing time \( f[v] \).

Example: CS courses

Why does topological sort work?

• Thm: Let \( G \) be a DAG. If \( G \) contains an edge \( u \to v \), then after running DFS, \( f[v] < f[u] \).
• (Transitively, this means that all vertices are correctly ordered by reverse finishing time.)

• For every edge \((u, v)\), when DFS traverses this edge . . .

• If \(v\) is undiscovered, it will be started and finished before returning to \(u\), so \(f[v] < f[u]\).

• If \(v\) is finished, it was finished before we started to expand \(u\), so \(f[v] < f[u]\).

• If \(v\) is in-progress, we have a cycle! Won’t happen in a DAG. QED