Lecture 4
Graphs
• Traversals
• DFS
• Acyclicity

Sofya Raskhodnikova
Depth-first search: alternate presentation
Traversals as generic templates

DFS and BFS are useful generic “templates” for graph algorithms

- **Modifying BFS:**
  - Bipartiteness (2-coloring)
  - Shortest paths (Dijkstra)
  - Minimum spanning trees (Prim)

- **Modifying DFS**
  - Finding cycles
  - Topological sort
  - Strongly connected components
DFS: setting up notation

- Maintain a global counter **time**
- Maintain for each vertex $v$
  - Two timestamps:
    - $v.d =$ time first discovered
    - $v.f =$ time when finished
  - “color”: $v.color$
    - **white** = unexplored
    - **gray** = in process
    - **black** = finished
  - Parent $v.\pi$ in DFS tree
**DFS pseudocode**

DFS \((G(V,E))\)

for each \(u \in V\)

\(u.color \leftarrow \text{WHITE}\)

\(u.\pi \leftarrow \text{NIL}\)

\(time \leftarrow 0\)

for each \(u \in V\)

if \(u.color = \text{WHITE}\)

\(\text{DFS-Visit}(G,u)\)

---

**Note:** recursive function different from first call...

---

DFS-Visit \((G,u)\)

\(time \leftarrow time + 1\)  // White vertex \(u\) is discovered

\(u.d \leftarrow time\)

\(u.color \leftarrow \text{GRAY}\)

for each \(v \in G.\text{Adj}[u]\)

if \(v.color = \text{WHITE}\)  // Explore edge \((u,v)\)

\(v.\pi \leftarrow u\)

\(\text{DFS-Visit}(G,v)\)

\(u.color \leftarrow \text{BLACK}\)  // Finish exploring \(u\)

\(time \leftarrow time + 1\)

\(u.f \leftarrow time\)
DFS example, animated
Parenthesis Theorem

- If we represent $v.d$ and $v.f$ as matching open and closed parentheses (or brackets), so that each vertex gets its own type (or color) of parentheses, then the history of discoveries and finishings is a properly nested expression of parentheses.

- Specifically, for all $u, v \in V$,
  - the intervals $[u.d, u.f]$ and $[v.d, v.f]$ either
    - entirely disjoint (and $u, v$ are not descendant/ancestor)
    - or one of the intervals is entirely contained in the other
      (the interval of the descendant is contained in the interval of the ancestor)
DFS Edge Types

- Tree
- Forward
- Backward
- Cross

Classifying edges according to type gives info about graph structure
• T = tree edge
• F = forward edge (to a descendant in DFS forest)
• B = back edge (to an ancestor in DFS forest)
• C = cross edge (goes to a vertex that is neither ancestor nor descendant)
Modifying DFS to classify edges

- We have enough information to classify edges as DFS explores them

When \((u, v)\) is first explored:

- If \(v\) is WHITE, then \((u, v)\) is a tree edge
- If \(v\) is GRAY, then \((u, v)\) is a back edge
- If \(v\) is BLACK, then \((u, v)\) is a forward or cross edge

Exercise: Show that in this case,

- if \(u.d < v.d\) then \((u, v)\) is a forward edge;
- if \(u.d > v.d\) then \((u, v)\) is a cross edge.
Running time with adjacency lists

- **Outer code** runs once, takes time $O(n)$ (not counting time for recursive calls)
- **Recursive calls:**
  - Run once per vertex
  - time = $O(\text{degree}(v))$

```
DFS(G(V,E))
for each u ∈ V
    u.color ← WHITE
    u.π ← NIL
time ← 0
for each u ∈ V
    if u.color = WHITE
        DFS-Visit(G,u)
```

```
DFS-Visit(G,u)
time ← time + 1 // u is discovered
u.d ← time
u.color ← GRAY
for each v ∈ G.Adj[u] // Explore edge (u,v)
    if v.color = WHITE
        v.π ← u
        DFS-Visit(G,v)
u.color ← BLACK // Finish exploring u
time ← time + 1
u.f ← time
```

- $\sum_v \text{degree}(v) = m$ or $2m$
- **Total:** $O(m + n)$
Review Questions

• Suppose we run DFS on a directed graph G.

• True or false?
  1. G has a cycle if and only if there exists a back edge
  2. G has a cycle if and only if there exists any non-tree edges
  3. G has a cycle if and only if there exists a forward edge

Exercise: Write a proof of the true statements; give counterexamples for false ones.