CSC 505, Fall 2000: Week 8

Objectives:

• learn about the basic depth-first search algorithm
• learn how properties of a graph can be inferred from the structure of a DFS tree
• learn about one nontrivial application of DFS to obtain a linear-time graph algorithm
Depth-first search: a simple example
**DFS: the algorithm**

**procedure** DFS(\(G\))  \hspace{1em} is
\>
> “visits” all vertices of the graph

for each vertex \(u \in V[G]\) do \(color[u] \leftarrow \text{WHITE}; \pi[u] \leftarrow \text{NIL}\)

\(time \leftarrow 0\)

for each vertex \(u \in V[G]\) do
\>
> if \(color[u] = \text{WHITE}\) then DFS-Visit(\(u\))

end DFS

**procedure** DFS-Visit(\(u\))  \hspace{1em} is

\(\text{color}[u] \leftarrow \text{GRAY}; \text{d}[u] \leftarrow \text{time} \leftarrow \text{time} + 1\)

\(\triangleright \) Insert any “beginning of visit” code here.

for each vertex \(v \in \text{adj}[u]\) do
\>
> \(\triangleright \) Explore edge \(uv\)
\>
> \(\triangleright \) Insert code to process edge \(uv\) here
\>
> if \(color[v] = \text{WHITE}\) then \(\pi[v] \leftarrow u\); DFS-Visit(\(v\))

\(\text{color}[u] \leftarrow \text{BLACK}; \text{f}[u] \leftarrow \text{time} \leftarrow \text{time} + 1\)

\(\triangleright \) Insert any “end of visit” code here.

end DFS-Visit
Edge classification by DFS

During DFS-Visit( \( u \) ), \( v \) is

- **white**: \( d[u] < d[v] < f[v] < f[u] \)
  - tree edge
- **gray**: \( d[v] < d[u] < f[u] < f[v] \)
  - back edge (edge to an ancestor in the tree)
- **black**: \( d[v] < f[v] < f[u] \)
  - cross edge if \( d[v] < d[u] \)
  - forward edge if \( d[v] > d[u] \)

In general, \( u \) is an ancestor of \( v \) iff \( d[u] < d[v] < f[v] < f[u] \);
\( u \) is unrelated to \( v \) if \( d[u] < f[u] < d[v] < f[v] \) or vice-versa.
Cycles and back edges

Lemma 8.1. A graph $G$ has a cycle if and only if every DFS of $G$ has a back edge.

Proof. (back edge $\Rightarrow$ cycle) In order for $uv$ to be a back edge, there must be a path of tree edges from $v$ to $u$ (since $v$ is an ancestor of $u$).

(cycle $\Rightarrow$ back edge) Let $f$ be the finish times of an arbitrary DFS and let $C$ be a cycle of $G$. Let $u$ be the vertex with minimum $f$ in $C$ and let $v$ be the vertex after $u$ in $C$. Since $f[u] < f[v]$, $uv$ must be a back edge (all other kinds have $f[u] > f[v]$).
A **directed acyclic graph** (dag) is a directed graph with no cycles. A **topological sort** of a dag is a permutation of the vertices in which every edge of the dag goes from an earlier vertex to a later vertex (a numbering for which every edge goes from a lower to a higher number).

**Observation:** Any permutation derived from a DFS sorting vertices by decreasing finish time is a topological sort.

The converse is also true: You can always access the vertices in the outer loop of DFS in reverse order of any topological sort and get that same permutation back.
**Strong components**

Two vertices $u$ and $v$ are in the same strong component of a directed graph if there is a path from $u$ to $v$ and a path from $v$ to $u$.

A directed graph whose vertices are all in the same strong component is strongly connected.

The dag of strong components.
Some questions

1. Can a DFS forest of a strongly connected graph have more than one tree?

2. If there is only one tree in a DFS forest, is the graph strongly connected?

3. If two vertices are in the same strong component, are they in the same tree of any DFS?
Forefathers

The forefather of vertex $u$ with respect to a DFS tree, denoted $ff(u)$, is a vertex $v$ such that (a) there is a path from $u$ to $v$, and (b) $f[v]$ is maximum among all $v$ that are “reachable” from $u$ by a path.
**Forefather \( \Rightarrow \) ancestor**

**Lemma 8.2.** If \( v = fi(u) \) then \( v \) is an ancestor of \( u \) in the DFS tree.

**Proof.** If \( v = fi(u) \) then \( f[v] \geq f[u] \) (since \( u \) has a path to itself). There are three possibilities.

- **different trees**
- **different branches**
- **ancestor**
Same forefather \(\iff\) same strong component

**Theorem 8.3.** Vertices \(x\) and \(y\) have the same forefather in a DFS tree \iff they are in the same strong component.

**Proof.**

*Part 1.* \(\text{ff}(x) = \text{ff}(y) \implies x\) and \(y\) are in the same strong component. Suppose \(\text{ff}(x) = \text{ff}(y) = v\). Then \(x \sim v\) and \(y \sim v\) by definition of \(\text{ff}\). Also, \(v \sim x\) and \(v \sim y\) by Lemma 8.2. Thus \(x \sim y\) and \(y \sim x\) as desired.

*Part 2.* \(x\) and \(y\) in the same strong component \(\implies \text{ff}(x) = \text{ff}(y)\). Let \(v_x = \text{ff}(x)\) and \(v_y = \text{ff}(y)\). If \(v_x \neq v_y\) assume \(f[v_x] < f[v_y]\). But \(x \sim y\) and \(y \sim v_y\), meaning \(x \sim v_y\), which contradicts the fact that \(v_x = \text{ff}(x)\). \(\square\)
Consider the vertex $v$ with latest finish time (largest $f[v]$) after a DFS.

**Q:** Who can reach $v$?

**A:** Exactly the vertices whose forefather is $v$?

**Q:** How do we find these vertices?

**A:** Do a DFS-Visit from $v$ on $G^R$, the graph with all edges reversed.

After eliminating these vertices, we have reduced to an instance have one fewer strong component (and at least one less vertex). Repeat the above process starting with the remaining vertex of latest finishing time.

The algorithm takes $\Theta(m + n)$, as does any algorithm based on a constant number of applications of DFS.
The algorithm on an example