Lecture 8: DFS and Topological Sort
CLRS 22.3, 22.4

Outline of this Lecture

• Recalling Depth First Search.

• The time-stamp structure.

• Using the DFS for cycle detection.

• Topological sort with the DFS.
What does DFS Do?

Given a digraph $G = (V, E)$, DFS traverses all vertices of $G$ and

- constructs a forest, together with a set of source vertices; and

- outputs two time unit arrays, $d[v]/f[v]$.

**DFS Forest:** DFS constructs a forest $F = (V, E_f)$, a collection of trees, where

$$E_f = \{(pred[v], v)| \text{ where DFS calls are made}\}$$
A Depth First Search Example

Example

Question: What can we do with the returned data?
Classification of the Edges of a Digraph

Tree edges: those on the DFS forest.

The remaining edges fall in three categories:

Forward edges: \((u, v)\) where \(v\) is a proper descendant of \(u\) in the tree. \([u \neq v]\)

Back edges: \((u, v)\), where vertex \(v\) is an ancestor of \(u\) in the tree. \([u = v\) is allowed.]

Cross edges: \((u, v)\) where \(u\) and \(v\) are not ancestors or descendants of one another (in fact, the edges may go between different trees).
Example of the Classification of Edges

Remark: Since the forest obtained with the DFS is not unique, the classification is not unique.
There is also a nice structure to the time stamps, which is referred to as *Parenthesis Structure*.

**Lemma 1** Given a digraph $G = (V, E)$, any DFS Forest for $G$, and any two vertices $u, v \in V$,

- $u$ is a descendent of $v$ *in the DFS forest* if and only if $[d[u], f[u]]$ is a subinterval of $[d[v], f[v]]$: $d[v] < d[u] < f[u] < f[v]$  

- $u$ is *unrelated* to $v$ *in the DFS forest* if and only if $[d[u], f[u]]$ and $[d[v], f[v]]$ are disjoint intervals: $f[u] < d[v]$ or $f[v] < d[u]$  

Cycles in digraphs: Applications of DFS

Claim: Given a digraph, DFS can be used to determine whether it contains any cycles.

Lemma 2: Given a digraph $G$, and any DFS tree of $G$, tree edges, forward edges, and cross edges all go from a vertex of higher finish time to a vertex of lower finish time. Back edges go from a vertex of lower finish time to a vertex of higher finish time.

Proof: The conclusions follow from Lemma 1.
When Is a Digraph Acyclic

Lemma 3: A digraph is acyclic if and only if any DFS forest of $G$ yields no back edges.

Example:

Original graph:

```
original graph
```

C: cross, F: forward, B: back edge
When Is a Digraph Acyclic

**Lemma 3:** A digraph is acyclic if and only if any DFS forest of $G$ yields no back edges.

**Proof of $\iff$:** Suppose there are no back edges. By Lemma 2, all edges go from a vertex of higher finish time to a vertex of lower finish time. Hence there can be no cycles.
Lemma 3: A digraph is acyclic (a DAG) if and only if any DFS forest of $G$ yields no back edges.

Proof of $\Rightarrow$: Assume that $G$ has no cycles. We prove that $G$ has no back edges by contradiction.

Suppose there is a back edge $(u, v)$. Then $v$ is an ancestor of $u$ in a rooted DFS tree. There is a path $v \rightarrow \ldots \rightarrow u$ in the DFS tree.

The back edge + the path gives a cycle. A contradiction!
Cycle Detection with the DFS

**Cycle detection:** becomes back edge detection by Lemma 3!

**Problem:** Modify the DFS algorithm slightly to give an algorithm for cycle detection.
This can always be done by first running the algorithm and assigning the $d[v]$ and $f[v]$ values and then running through all of the edges one more time, seeing if any of them are back edges. This would take $O(n + e)$ time for the DFS and $O(e)$ time for the scan through all of the edges. In total, this uses $O(n + e)$ time.

How could you solve this problem *online* by identifying back edges while the DFS algorithm is still running?
If $G = (V, E)$ is a DAG then a topological sorting of $V$ is a linear ordering of $V$ such that for each edge $(u, v)$ in the DAG, $u$ appears before $v$ in the linear ordering.

**Example:** Let $V = \{1, 3, 4, 5, 6, 12\}$ and have $(a, b) \in E$ if and only if $a | b$. This is partial order, but not a linear one.

There are several topological sortings of $G$ (how many?), for example:

$\langle 1, 3, 4, 5, 6, 12 \rangle$, $\langle 1, 4, 3, 6, 12, 5 \rangle$, $\langle 1, 5, 3, 6, 4, 12 \rangle$. 
If $G = (V, E)$ is a DAG then a topological sorting of $V$ is a linear ordering of $V$ such that for each edge $(u, v)$ in the DAG, $u$ appears before $v$ in the linear ordering.

**Idea of Topological Sorting:** Run the DFS on the DAG and output the vertices in reverse order of finishing time.

**Correctness of the Idea:** By lemma 2, for every edge $(u, v)$ in a DAG, the finishing time of $u$ is greater than that of $v$, as there are no back edges and the remaining three classes of edges have this property.
Topological Sort: the Algorithm

The Algorithm:

1. Run DFS(G), computing finish time $f[v]$ for each vertex $v$;

2. As each vertex is finished, insert it onto the front of a list;

3. Output the list

Running time: $\Theta(n + e)$, the same as DFS.
Topological Sort: Example

Original graph  DFS forest

Final order: \( \langle b, f, g, a, c, d, e, h \rangle \).