6.006 Lecture 14: DFS & Topological Sort

- DFS
- DFS classifies edges in a useful way
- Directed Acyclic Graphs
- Topological sort
- Scheduling parallel computations

CLRS 22.3, 22.4
Generic Graph search from last time:

\[ R = \text{set} ([s]) \]
\[ Q = \text{set} ([s]) \]

while \( Q \) is not empty:

\[ u = \text{Q. dequeue} \quad \text{# can be LIFO, FIFO, etc.} \]

for \( v \) in \( \text{Adj}[u] \):
    if \( v \) not in \( R \):
        \[ R. \text{add}(u) \]
        \[ Q. \text{add}(v) \]

DFS (Depth-First Search)

```python
def visit(u):
    for v in \text{Adj}[u]:
        if v not in R:
            R.\text{add}(v)
            visit(v)
```

\[ R = \text{set}() \]

for \( u \) in \( V \):
    if \( u \) not in \( R \):
        \[ R.\text{add}(u) \]
        \[ \text{visit}(u) \quad \text{# search entire graph} \]

Where is \( Q \)?
DFS Example:

\[ V = \{a, b, c, d, e, f\} \]
\[ \text{Adj}[a] = \{b, d\} \]

- \( \rightarrow \) \( v \) not in \( R \)
- \( \Rightarrow \) \( v \) in \( R \)

Classification of edges:

- Tree edges, \( u \) to a child \( v \) (1, 3, 5, 6)
- Back edges, \( u \) to an ancestor (4)
- Forward edges, \( u \) to a descendant (5)
- Cross edges, \( u \) to another subtree (5)
- Self loops (7)
Acyclic Graphs \equiv \text{DAG} \text{ directed}

\[ \forall G \text{ is a DAG iff } G \text{ contains no cycles.} \]

Thm: $G$ is a DAG $\iff$ DFS of $G$ produces no back edges or self loops

Proof: $\implies$ if there is a back edge $u \rightarrow v$ or a self loop there is a cycle, contradiction

$\iff$ we say that $v$ "finishes" when $\text{visit}(v)$ returns.

Lemma: in DFS if $(u, v)$ is not a back edge or self loop then $v$ finishes before $u$

Proof: Tree edge: $\text{visit}(v)$ calls $\text{visit}(u)$, so $\text{visit}(u)$ finishes first.

Forward edge: $\text{visit}(u)$ already returned (since we are back in $\text{visit}(u)$)

Cross edge: $\text{visit}(u)$ returned.

End of proof of theorem: if there was a cycle, by following its edges we would get earlier & earlier finishing times, which is not possible.
Checking if a graph is acyclic

Idea: augment DFS data structure to detect
back edges & self loops

for v in V: color[v] = white  # initialization; not in R

def visit(u):
    color[u] = gray  # in R and in Q
    for v in Adj[u]:
        if color[v] = gray: G is cyclic
    if v not in R:
        R.add(u)
        visit(v)
    color[v] = black  # in R, not in Q
Topological Sort

\[ V = \text{set of tax form to fill in (just an example)} \]
\[ U \rightarrow V \text{ means you must fill } U \text{ in before you can fill } V \text{ in} \]

Topological sort: a feasible order for filling in all forms

\[ V = [A, B, C, D, E, F, G, H, I] \]

Output: I G D E A H B C F

Is the order unique?
Parallel Processing

If you had many helpers, how quickly could you fill in all the tax forms? (The helpers can work in parallel)

In the example: 4 steps, critical path (longest path in a DAG) is A → B → C → F.

Idea: produce topological sort, take an element at a time, put in earliest possible time slot

Time: 1 2 3 4

I → E → C → F
G → H → D → B → A