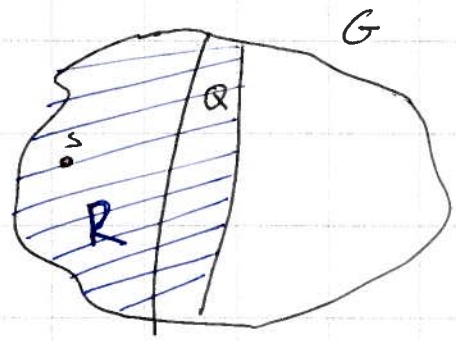
6.006 Lecture 14: DFS & Topological Sort

- 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 = set([s])
Q = set([s])
while Q is not empty:
    u = Q.dequeue          # can be LIFO, FIFO, etc
    for v in Adj[u]:
        if v not in R:
            R.add(v)
            Q.add(v)
```

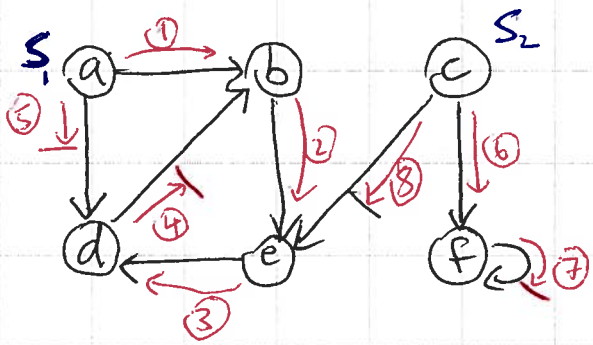


DFS (Depth-First Search)

```
def visit(u)
    for v in Adj[u]:
        if v not in R:
            R.add(v)
            visit(v)

R = set()
for u in V:
    if u not in R:
        R.add(u)
        visit(u)          # search entire graph
```

Where is Q?

DFS Example:



$V = [a, b, c, d, e, f]$

$Adj[a] = [b, d]$

⋮

→ v not in R

→| v in R

Classification of edges:

Tree edges, u to a child v (1)(2)(3)(6)

non-tree edges:

- Back edges, u to an ancestor (4)
- Forward edges, u to a descendant (5)
- Cross edges, u to another subtree (8)
- Self loops (7)

Acyclic Graphs ≡ DAG
directed

□ G is a DAG iff G contains no cycles.

<u>Thm</u>: G is a DAG ⟺ DFS of G produces no back edges or self loops

<u>Proof</u>: ⇒ if there is a back edge $u \to v$ or a self loop there is a cycle, contradiction

⇐ we say that v "finishes" when visit(v) returns.

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

<u>Proof</u>: Tree edge: visit(u) calls visit(v), so visit(v) finishes first.

forward edge: visit(v) already returned (since we are back in visit(u))

cross edge: visit(v) 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

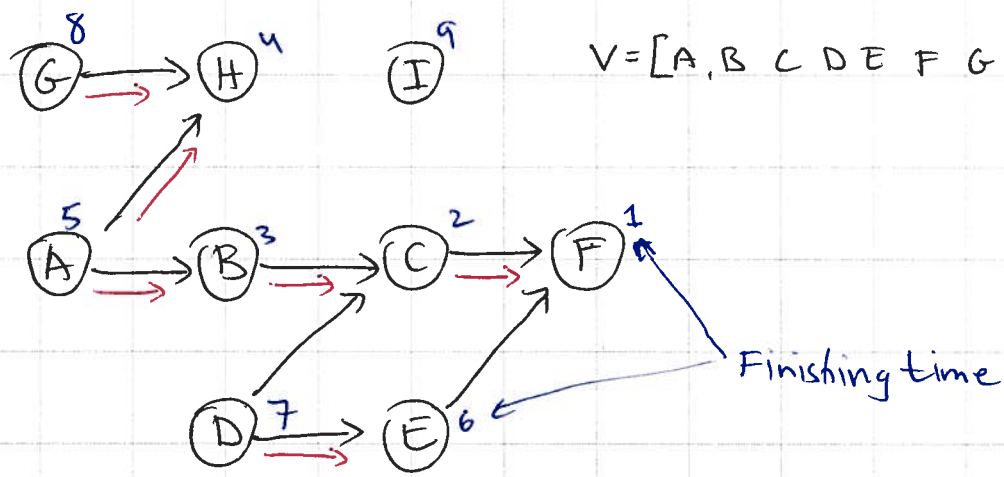
```
for v in V: color[v] = white        # initialization; not in R

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(v)
            visit(v)
    color[v] = black                # in R, not in Q
```

Topological Sort

V = set of tax form to fill in (just an example)

$u \to v$ means you must fill u in before you can fill v 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

all edges go from left to right

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 $A \to B \to C \to F$.

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

Time: