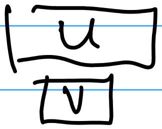
Depth First Search

```
Pre order = []
def previsit(v):
    preordr. append(v)
```

```
def DFS(v):
    visited. add(v)

    Previsit (v)

    for w in v.adj:
        if w not in visited:
            w. parent = v
            DFS(w)
    Post Visit (v)


def DFS All (G):
    for v in G:
        if v not in visited:
            DFS (v)
```

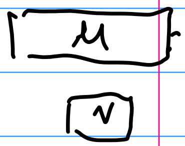
Edge $(u \to v) \in G$ can be:

① Tree edge (green):



$u.pre < v.pre < v.post < u.post$
& DFS calls v from u directly

② Forward edges (blue):

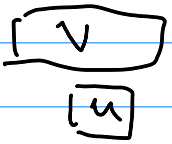


other edges w/

$u.pre < v.pre < v.post < u.post$

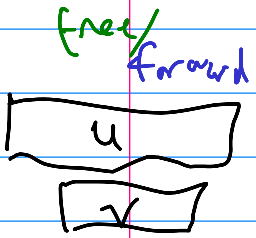
[Note: ∃ path $u \rightsquigarrow v$ along tree edges]

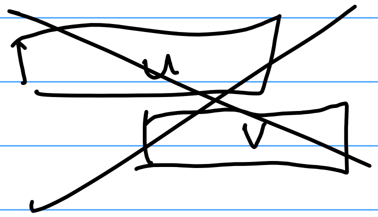
③ backward edges (red)

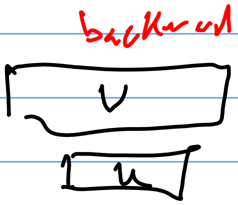


$v.pre < u.pre < u.post < v.post$

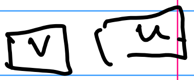
[Note: back + tree forms a cycle, ∃ $v \rightsquigarrow u$ path along tree edges]



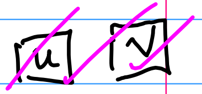




④ Cross edges (gray)



$v.pre < v.post < u.pre < u.post$



~~$u.pre < u.post < v.pre < v.post$~~
[u would call v]

Lemma $\forall$ u, v

u.pre < v.pre < v.post < u.post

$\Leftrightarrow$ u is ancestor of v in the DFS Forest

---

(1) Detect cycles

[if cycle exists, find it]

Lemma: G has a cycle $\Leftrightarrow$ $\exists$ backward edge

Pf Consider the reverse Postorder of G.

(u before v $\Leftrightarrow$ u.post > v.post)

Green, blue, gray all point forward

no red $\Rightarrow$ no cycle.

on the other hand,

$\exists$ red edge $u \to v$,

when you visit u,

v.pre < current time < v.post

$\Rightarrow$ stack contains $v \rightsquigarrow u$ path

$\Rightarrow$ that + $(u \to v)$ is a cycle.

(2) topological sort of DAG

find an order s.t.
all edges go forward

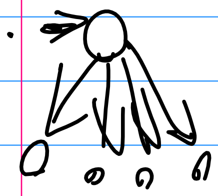
Reverse postorder

(3) *implicit* topological ordering:

visiting vertices in postorder
⟺ visiting in reverse topological
order.

Example longest $(s \rightsquigarrow t)$
path in DAG.

```
def longest(v, t):

    if v = t:  return 0
 -> if v in memo: return memo[v]
    memo[v] = max( longest(u, t)
                   + dist(v->u)
                 for u in v.adj)
    return memo[v]
```



Strong Connectivity

u & v are Strongly Connected
if u can reach v
& v can reach u.

Form Strongly Connected Components

& Strong Component graph

$SCC(G)$

which is a DAG
where vertices of $SCC(G)$
correspond to strong components of G.

Q: How to find Strong Components?

(a) For fixed v, find its strong component.

- For all u, see if
  v reaches u — whatever
  & u reaches v — first search

$O(V \cdot E)$

$\uparrow$ V other u, E time per u to search.

reach(v) = set of u reachable in O(E) time

$reach^{-1}(v)$ = set of u reachable on reversed graph

$\Rightarrow$ answer : $reach(v) \cap reach^{-1}(v)$

(b) find <u>all</u> SCC

- Run (a) on any vertex you don't know component for, repeat
  $O(VE)$
- cleverly: $O(E+V)$

<u>Note</u> Each strong component has exactly one parent [in DFS tree) not in component

reverse postorder of rev(G)

<u>SCC alg</u>

(1) find some vertex in a sink component

(2) its component is anything it can reach

(3) remove & repeat.

time = # edges it can reach

= # edges removed

$O(E)$ total

Kosaraju:

the last vertex in a postorder is in a <u>source</u> SCC.

⇒ last vertex in postorder of rev(G) is in a <u>sink</u> SCC