CMSC 451 Sep. 9

Depth-First Search (DFS)

A systematic process for visiting the vertices and edges of a graph.

Intuition: vertex to vertex, backtracking only when all neighbors have been visited

Algorithm: $G = (V, E)$ digraph

mark[u] : undiscovered / visited / finished

d[u] : discovery time

f[u] : finish time

p[u] predecessor

```
DFSVisit(u)
    mark[u] ← visited
    d[u] ← ++time
    for each (u,v) ∈ E
        if mark[v] = undiscovered
            pred[v] ← u
            DFSVisit(v)
    mark[u] ← finished
    f[u] ← ++time
```

for $v \in V$, if $A[u,v]=1$

$\forall u \in V$, out-deg(u) such edges.

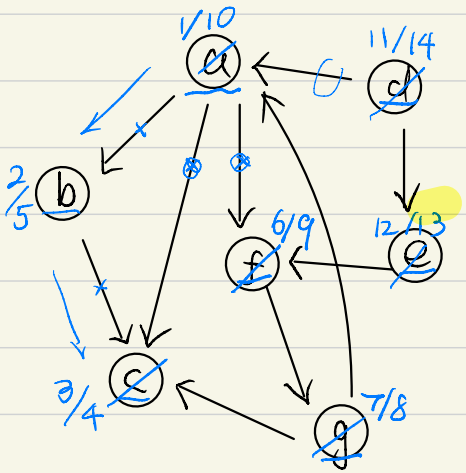
total: $\sum_u$ out-deg(u) $= |E|$

```
DFS(G)
    time ← 0
    mark[u] ← undiscovered.  ∀ u∈V
    for each u ∈ V
        if mark[u] = undiscovered
            DFSVisit[u]
```

Running time: $O(n+m)$ $n=|V|$ $m=|E|$

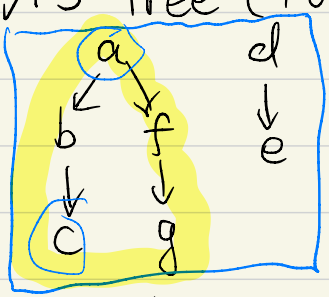
$O(n^2)$ adjacency matrix

Example



	a	b	c	d	e	f	g
pred	/	a	b	/	d	a	f
d	1	2	3	11	12	6	7
f	10	5	4	14	13	9	8

DFS Tree (Forest)



1 2 3 4 5 6 7 8 9 10 11 12 13 14

(a) (d)

(b) (f) (e)

(c) (g)

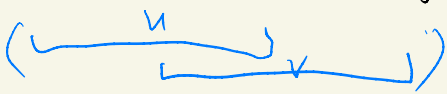
Parenthesis Lemma:

$\forall u, v \in V$ in DFS Tree:

u is descendant of v $\iff [d[u], f[u]] \subseteq [d[v], f[v]]$

ancestor $\supseteq$

otherwise $[d[u], f[u]]$ is disjoint from $[d[v], f[v]]$



no overlap.

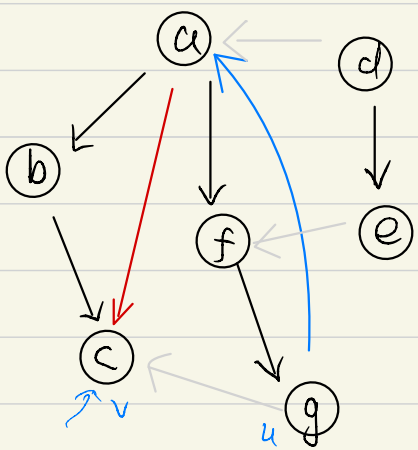
Edge Classification:

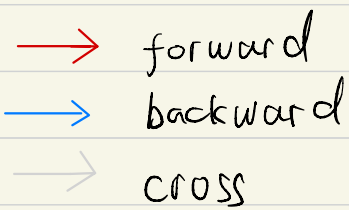
Tree edge: u discovers v

Back edge: v is an ancestor of u

Forward edge: v is a non-child descendant of u

Cross edge: anything else





- undirect graph: only tree & backward

- how to classify?

back edges $\iff f[u] \le f[v]$

back (u,v) $d[v] \le d[u] < f[u] \le f[v]$

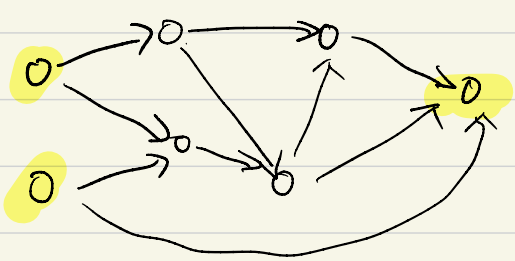
forward / tree $\rightarrow$ $d[u] \le d[v] \le f[v] \le f[u]$

tree $\leftarrow$ $pred[v] = u$

cross: $[d[u], f[u]]$ & $[d[v], f[v]]$ disjoint.

$d[v] \le f[v] < d[u] \le f[u]$

Directed Acyclic Graph (DAG)



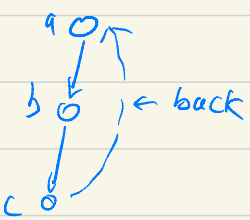
source : v. s.t. in-deg$(v) = 0$

sink : v. s.t. out-deg$(v) = 0$

Acyclicity Test:

Given a digraph G, is G a DAG?

Idea: Try DFS tree:



Back edge $\Rightarrow$ cycle

Cycle $\Rightarrow$ Back edge !

why?

> Lemma: (u,v) is back edge $\Leftrightarrow f[u] \leq f[v]$

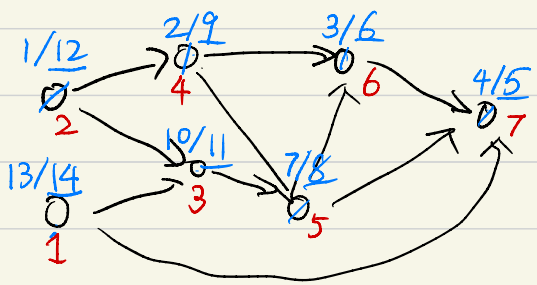
Assume cycle $a \to b \to c \cdots \to a$, no edges are back

$f[a] > f[b] > f[c] > \cdots > f[a]$ ⟶ contradiction.

Topological Sorting:

Given a DAG $G = (V, E)$, find a linear ordering S of the vertices, that respects edge directions, i.e.

$$(u, v) \in E \Rightarrow S(u) < S(v)$$



multiple ordering.

Idea: DAG $\Rightarrow$ no back edges

$\Downarrow$

$\forall (u,v) \in E,\ f[u] > f[v]$

$\Downarrow$

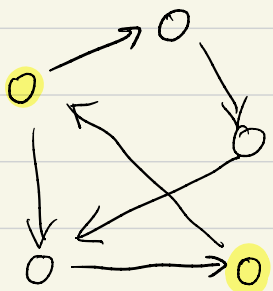
$S \leftarrow$ reverse order of f

Time: $O(n+m)$

Another algorithm: $O(n+m)$, implement carefully!
repeat: pick a source vertex
remove it.

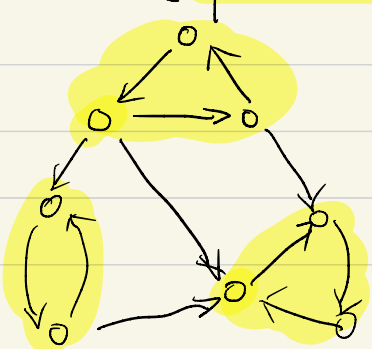
Strongly Connected Components (SCC)

Strongly connected: $\forall\ u, v \in V$, $\exists$ path $u \rightsquigarrow v$ & $v \rightsquigarrow u$.

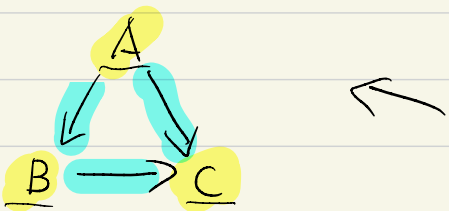


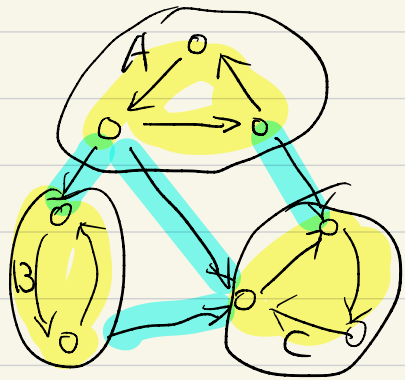
Strongly connected component (SCC):
A partition of vertices, s.t.

Two vertices u & v $\in$ same SCC $\iff$
$\exists$ path $u \rightsquigarrow v$ & $v \rightsquigarrow u$.

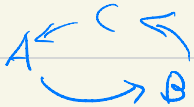


Component Digraph: "collapse" the SCC's





Claim: the component digraph is a DAG



$u \in A$, $v \in B$

$\exists$ path $u \to v \Rightarrow u, v \in$ same SCC

$v \to u$

Q: How to compute the SCCs?

Idea: DFS, starting at $u \in$ component X, visit all "downstream" SCCs.

DFS according to the reverse order (topological) of the component graph!

How to get the order (without knowing the SCCs)?

Recall: (u,v) back edge $\Leftrightarrow$ $f[u] \leq f[v]$

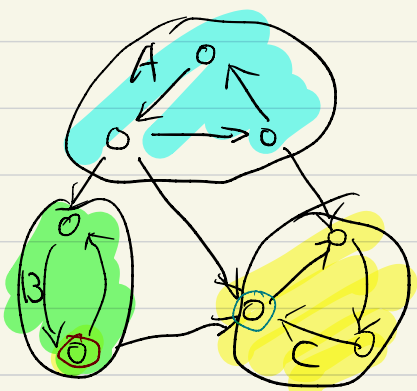
(u,v) across components $\Rightarrow$ $f[u] > f[v]$

Define max finish time $maxf[X]$ for SCC X.

$maxf[X] := \max_{v \in X} f[v]$.

Lemma: Let X, Y components, X can reach Y. (Y cannot reach X),

$maxf[X] > maxf[Y]$



Start $u \in A$. visit everything in V.

$u \in B$, ... $B \& C$

$u \in C$	C
$u \in B$	B
$u \in A$	A

last finished $v \in V$.
&
x, last finished

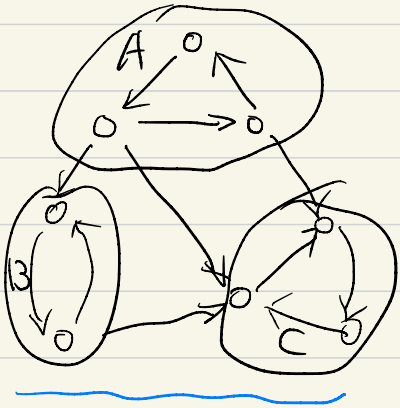
Algorithm (Kosaraju)

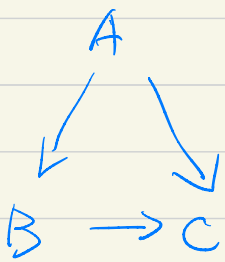
DFS(G). Compute $f[u]$, $\forall u \in V$.

DFS(G^R) in reverse order of $f[u]$.

Reverse Graph: reverse all edges.

$(u, v) \rightarrow (v, u)$





Reverse