Lecture 6, Paths in graphs

1. DAG

def. A directed acyclic graph (DAG)

Q: give digraph G, does G have cycles?

IsAcyclic(G):

run DFS on G, and output "DAG" if DFS reveals no **back edges** $O(n+m)$

def $(u,v) \in E$ is backedge if $[\ [\ ]\ ]$ $O(n+m)$ still

v u u v

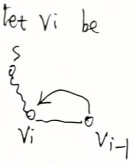
Claim: G is DAG if and only if DFS(G) reveals backedges.

Proof: back edge → cycle

cycle → back edge

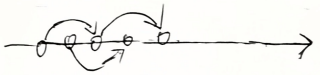
let $v_1, \dots, v_t$ be a cycle

let v_i be the first visited vertex.



Topological Sort

Given G that is a DAG, want to order the vertices in such a way that every edge points forward.



run DFS on G, output vertices in **descending post numbers**

Actually, DAG has no backedges

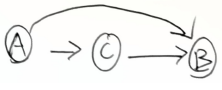
For forward, tree, cross edges, $u \to v$, u's postnumber is

$[\ [\ ]\ ]$ $[\ ]\ [\ ]$ often higher than v.

u v v u v v u u

A 1, 6

B 2, 3



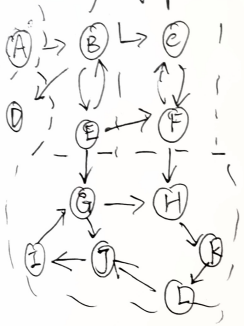
B←C C 4,5

Connectivity for digraphs.

def: u, v are connected if there is a path from u to v and from v to u. (strongly)

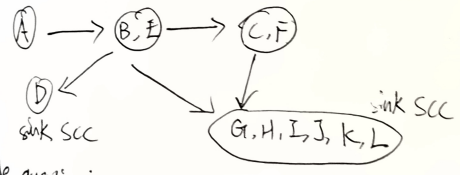
strongly connected components (SCC)

Claim: every digraph G is a DAG on SCCs.



Components: A, G, BE, CF, GHIJKL

sink SCC



sink SCC

sink SCC

side question:

given digraph G, find any v in source-SCC.

Answer: the vertex with ~~largest~~ ~~post~~ number.

SCC(G):

1. deduce G^R from G (source in G^R is sink in G)
2. run DFS on G^R to have post number
3. $\forall v \in V$ in reverse postorder of G^R, if not visited

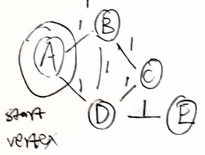
{explore((G), v). (not G^R)}

scc++

$v \in V$ in reverse post order of G^R is the topological sort

My understanding: find sink SCC by G^R (G^R's topological sort), use DFS to explore SCC from sink SCC. (sink SCC can't go to other SCC).

New topic: shortest paths. Q: find distance from A to all other vertices using BFS



A	B	C	D	E
0	1	2	1	2

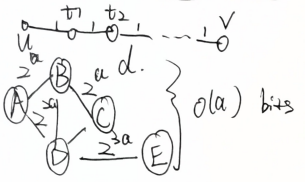

```
BFS (G, s)
    dist[s] = 0
    dist[V except s] = ∞
    Q = [s]
while Q ≠ []
    u = eject(Q)
    for (u,v) ∈ E
        if dist[v] = ∞
            inject(Q, v)
            dist[v] = dist[u] + 1
```

$T(n) = O(n+m)$

shortest paths with distances/weights.

idea: use BFS again by modifying graph.

Transform G into G' s.t. each edge $u \xrightarrow{d} v$ is replaced with



$T(n) = O(n'+m')$ $n' = |V'|$ $m' = |E'|$