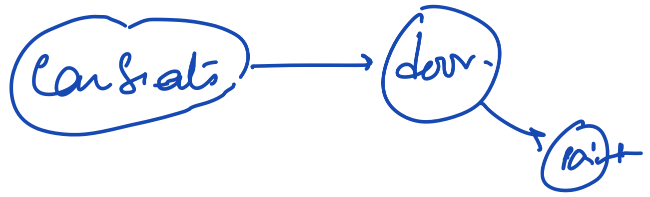
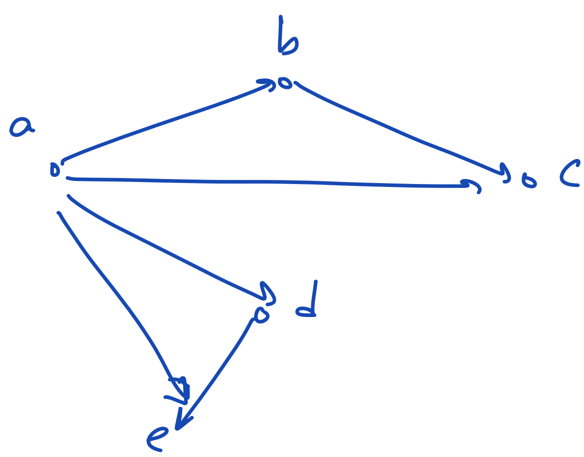
Topological Sort

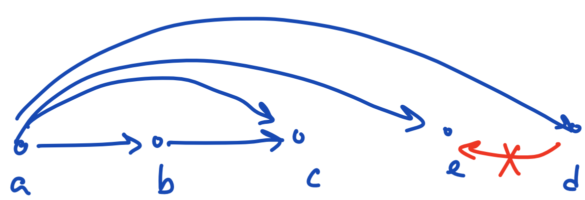


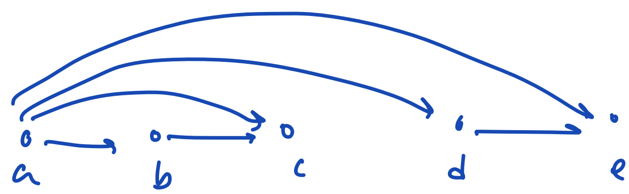
Input: Directed acyclic graph (DAG) $G = (V, E)$

Objective: Output a permutation of the vertices so that all edges go from left to right.

Topological Sort.





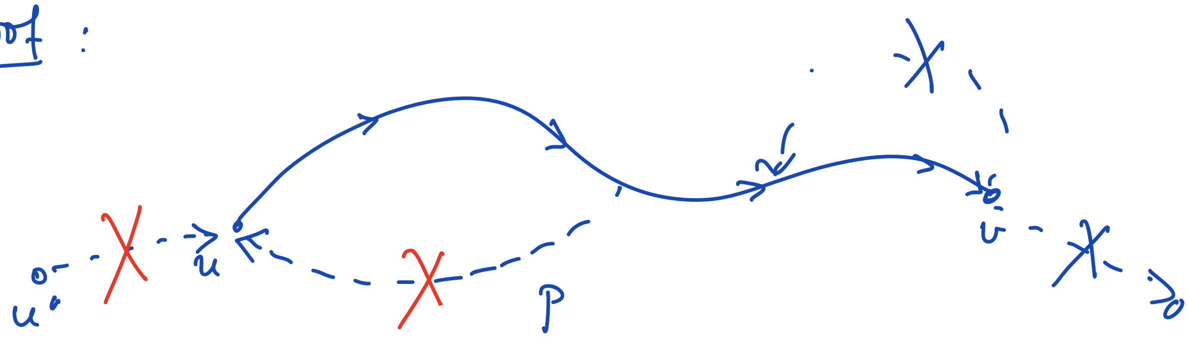


<u>Lemma</u> : Let $G = (V, E)$ be a DAG. Then G must have a <u>source</u> & a <u>sink</u>.

vertex with indegree 0 ✓

vertex with outdegree = 0

<u>Proof</u> :



Let P be a maximal path in G with end vertices u & v.

u : source. ✓

Similarly we can argue that

v is a sink.

TS (G)

1. $u \leftarrow$ a source vertex in G.

2. $G' \leftarrow G - u$

3. $\underline{L \leftarrow TS(G')}$

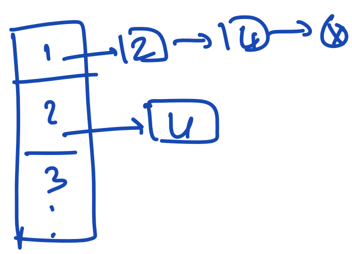
4. Output u followed by vertices in L.

n.
$n-1$
$n-2$
$\vdots$
1

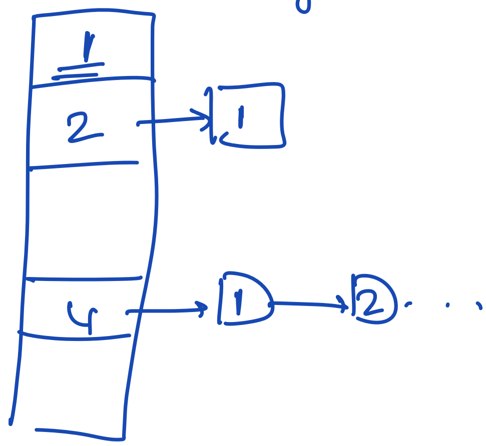
$\underline{n^2}$

Running time : $O(n+m)$

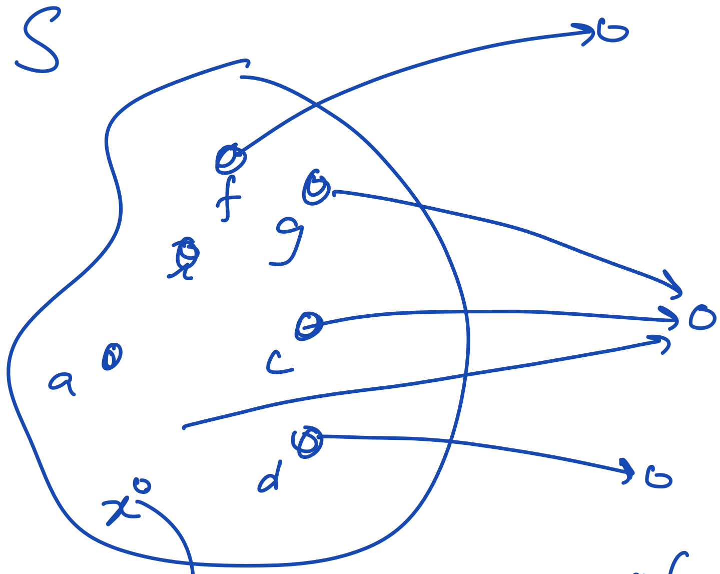
$\Omega(n^2)$.



Indegree list



Clever implementation.

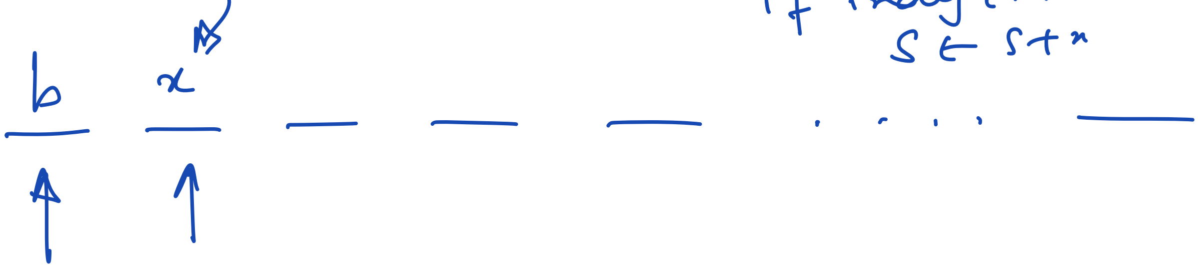


$$\sum_u dg(u) = O(m)$$

indg(x) --

if indg(x) = 0

$S \leftarrow S + x$



Alg

1. $S \leftarrow$ all sources in G $\leftarrow O(n+m)$

2. **while** $S \neq \phi$ **do**

3. $u \leftarrow$ any vertex in S

4. Append u to the output list L

5. **for** each $v \in N(u)$ **do**

$\sum_u \deg(u) = O(m$

6. indeg(v) --

"hi"

7. **if** indeg$(v) = 0$ **then**

8. $S \leftarrow S \cup \{v\}$

9. Output L

Running time : $O(n+m)$

Alternate solution (DFS based) ((Bob Tarjan))

Robert

Turing award 1970s.

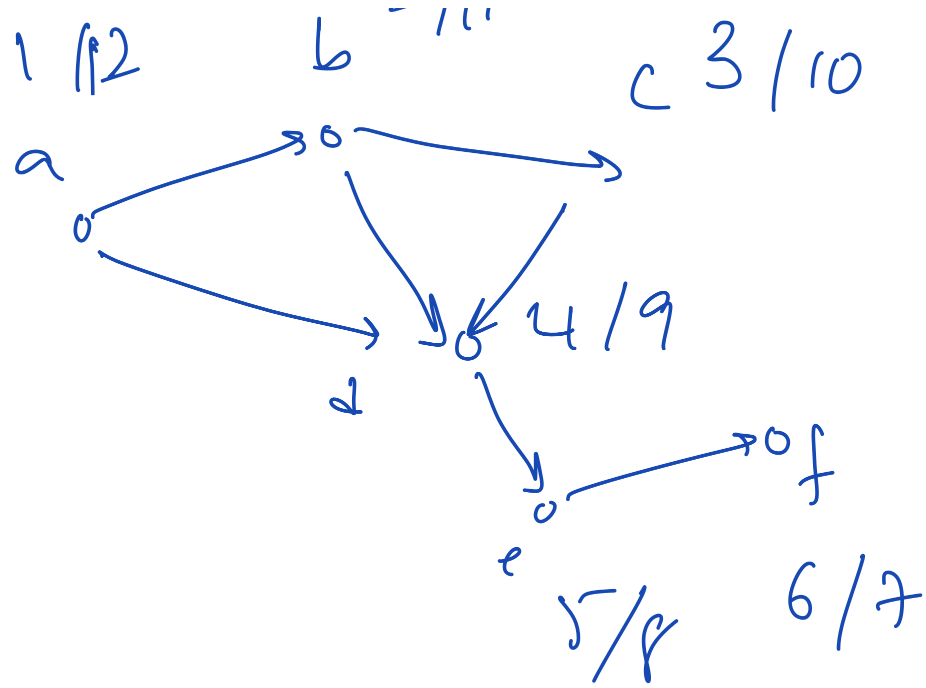
1. DFS(G). ← $O(n+m)$

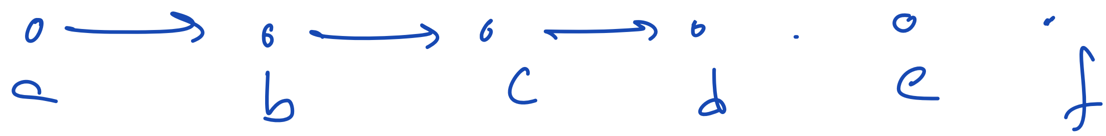
2. Sort vertices in decreasing order of $f[\cdot]$.

↳ $O(n \lg n)$

Running: $O(n \lg n + m)$ ✓

$O(n+m)$: while doing DFS, as vertices finish add them at the beginning of the list.





<u>Proof of Correctness</u>

<u>Thm</u>: Our alg. works.

<u>Proof</u>: Let $e = (u,v)$ be any edge in G.

We want to prove that u appears before v

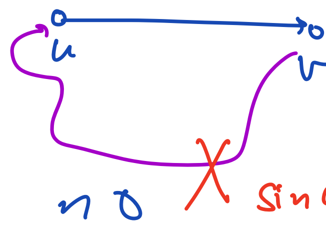
in the output. That is, to prove that $f[u] > f[v]$.

<u>Case I</u> : $d[u] < d[v]$ $\quad \boxed{u \longrightarrow v}$

At time $d[u]$ there is a white path from u to v in G. Thus, by the WPT, v is a descendant of u in the DFS forest. By the Parenthesis theorem, $d_u < d_v < \boxed{f_v < f_u}$.

Thus $f_u > f_v$

Case II : $d_u > d_v$.



since G is a DAG.

At time d_v there is no white path from v to u in G.

Thus, by WPT, u is <u>not</u> a descendant of v in the DFS forest. Thus, by the Parenthesis thm, we have

$\therefore f_u > f_v$ ✓.

Strongly Connected Components.

Input: Directed graph $G = (V, E)$

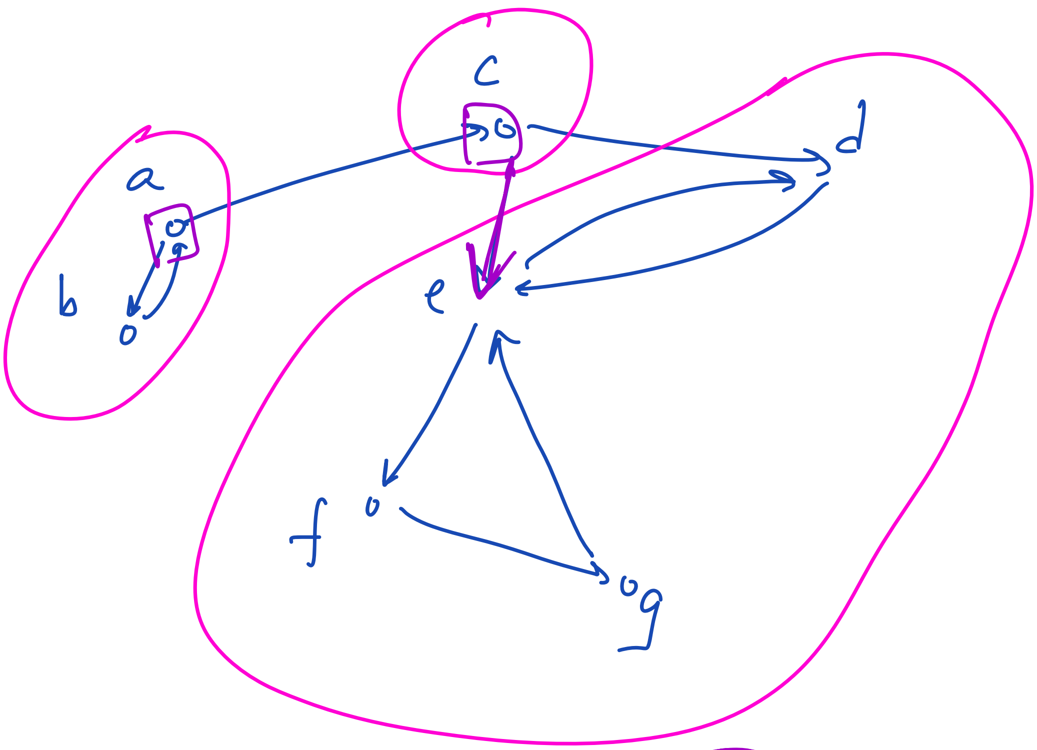
Objective: To output all strongly connected components (SCC) of G.

Defn: H is a strongly connected component of G if

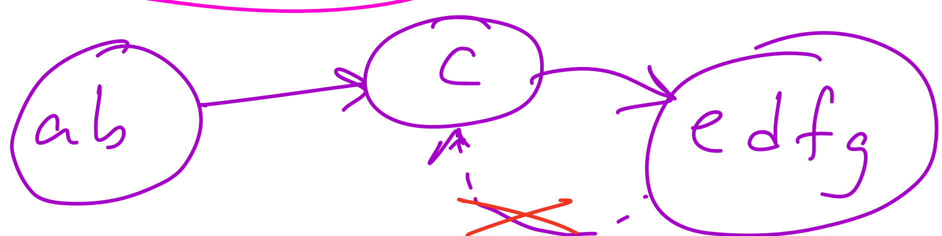
- H is subgraph of G
- $\forall\, u, v$ in H, $u \neq v$, there is a path from u to v in G and a path from

v ~~t~~ u in G.

- H is maximal.



G^{scc} :



Q Suppose we flip the directions of every edge in G then

SCCs in G remains the same.

Consider a graph $G^{scc} = (V^{scc}, E^{scc})$, in which each vertex in V^{scc} corresponds to a SCC in G. We have an edge $(C, C') \in E^{scc}$ iff there is an edge from a little vertex in C to a little vertex in C'.

Why is G^{scc} a DAG?

Otherwise the maximality property of a

SCC will be violated.