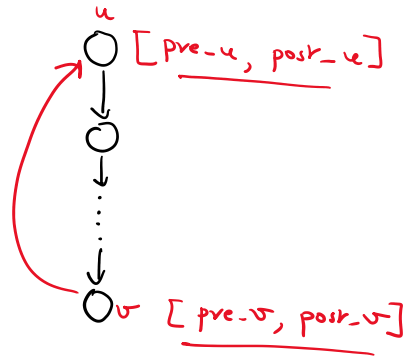
Directed Graphs

detecting cycles

→ linear time $O(|V|+|E|)$

→ pre-post numbers



forward / descendant edge

[ ]
↓
[ ]

backward edge

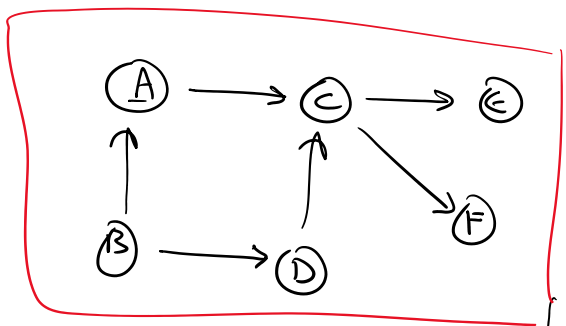
[ ]
↓
[ ]

Topological sorting

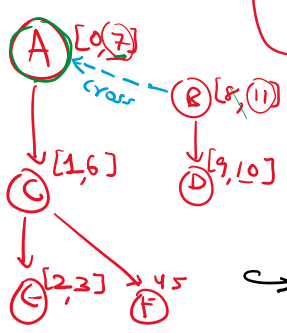
① acyclic directed graph ← $O(|V|+|E|)$

② order the nodes of the DAG in dependence order

Scheduling



pre-post from A

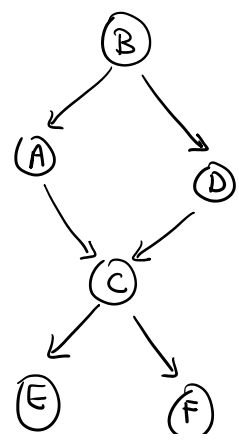


Topological sort

→ B A D C E F

B D A C F E

sorted order



B D A C F E sorted order

Alg 1 (G)

do along with step 1

1. DFS pre-post numbering $O(|V|+|E|)$
2. If Cycle detected, return NOT SORTABLE
3. output the sorted nodes in $\boxed{\text{decreasing order}}$ of post numbers $O(|V| \log |V|)$?

a) [for any node u, its post number must be greater than all other node that are reachable / descendants]

$\Rightarrow$ u must come before all such nodes in the topological sort

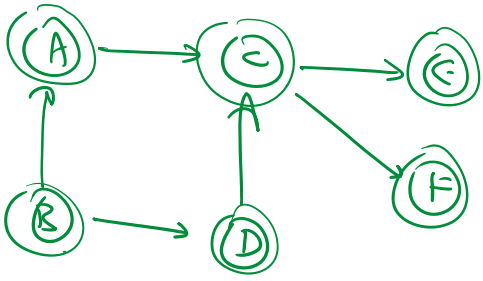
b) If u has an incoming edge it must be a cross edge from x since there are no cycles

[post_x > pre_x] > post_u

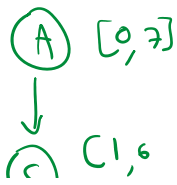
$\Rightarrow$ x has a higher post number than u

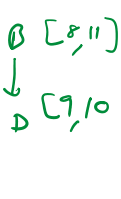
$O(|V|+|E|)$ time

↳ what about sorting by post numbers $O(|V| \log |V|)$ time ?



add the node to a list as we determine the post number

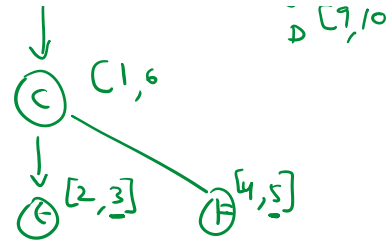




Post Order list

DFS time $O(|V| + |E|)$

Sorting! $O(|V|)$



F C A D B

reverse it!

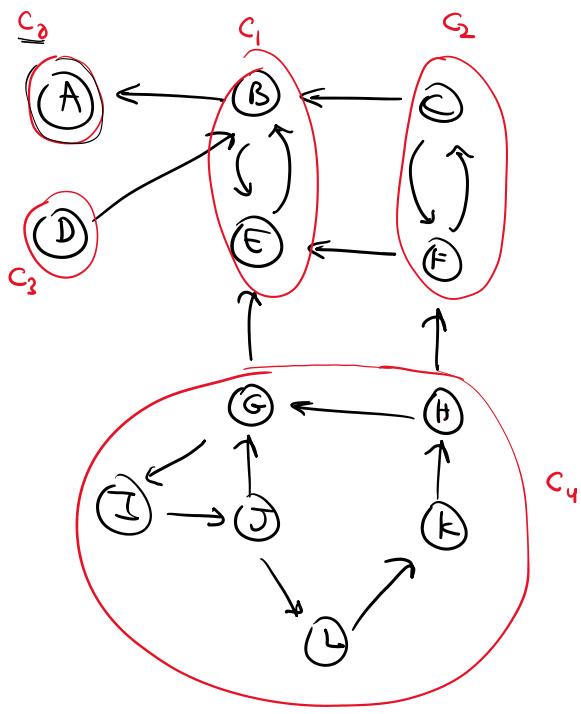
B D A C F E

Strongly Connected Components (SCCs)

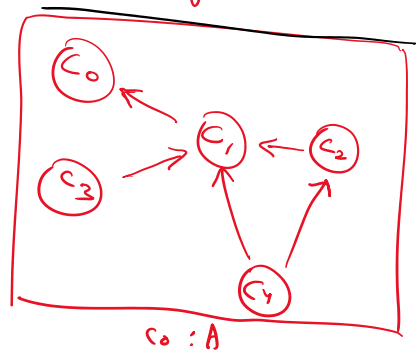
Directed graph: 1) a path must exist between all pairs of nodes in a SCC

2) it must be a maximal set

all mutually reachable node must be in SCC



DAG of SCCs



C_0 : A
C_1 : B E
C_2 : C F
C_3 : D
C_4 : G H I J K L

Any DAG must have at least

One source : no in edge

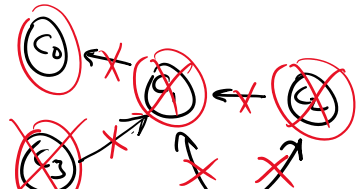
One sink : no out edge

Approach to list the SCCs

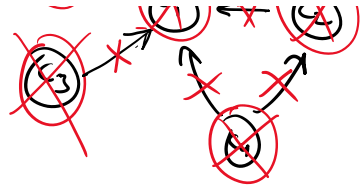
not good! → 1. find a sink SCC in G

2. delete that SCC

3. repeat.

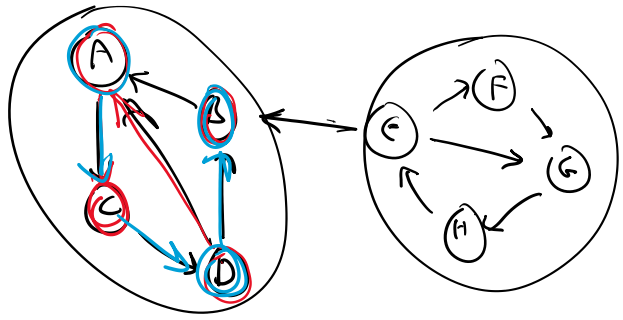


③ repeat.



C_3 C_4 C_2 C_1 C_0

finding sink components is not obvious



sink component

any DFS cannot escape a sink component

Any source in the reverse graph G^R is a sink in G

$O(|V|+|E|)$ 1) given G, compute reverse graph G^R, do DFS numbering on G^R

$O(|V|+|E|)$ 2) do connected components on G from the next unvisited vertex in reverse order of post-numbering

Computing G^R

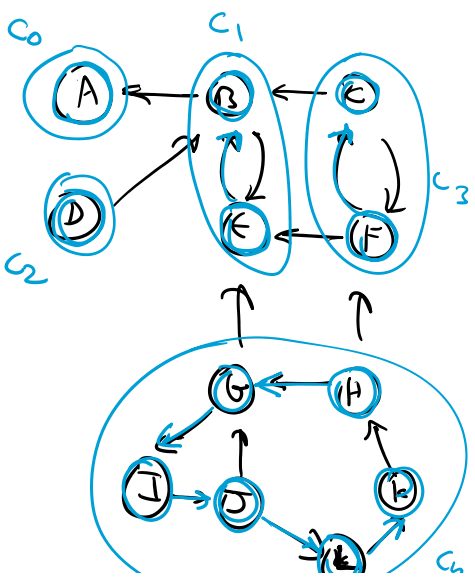
$O(|V|+|E|)$

overall

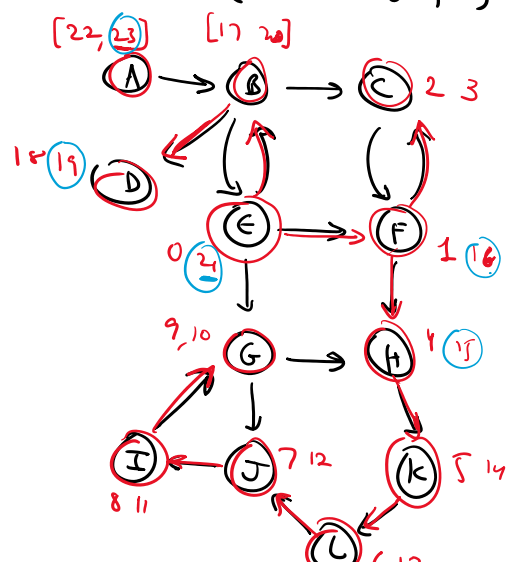
$O(|V|+|E|)$!

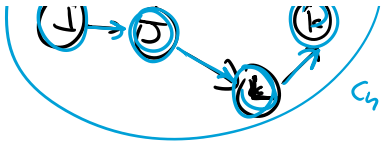
linear

G



G^R (reverse graph)

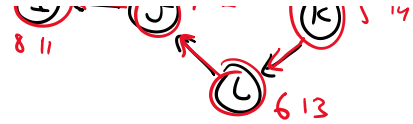




identify sinks in G; not clear how to do this

⇓

apply regular connected components since we cannot escape a sink



Just do DFS numbering

process the nodes in reverse order of the post numbers

reverse G into G^R; identify the source via largest post number

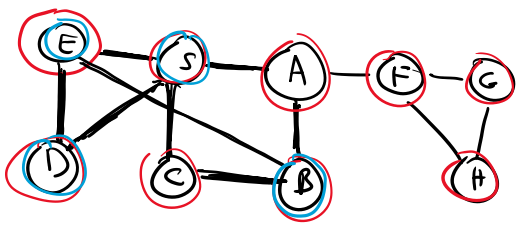
easy!

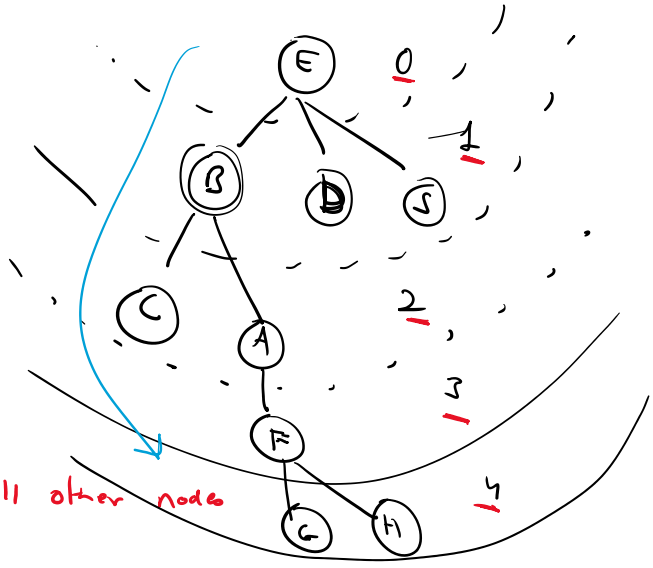
Breadth first search (BFS)

↳ weighted graphs.

↳ Shortest paths

Q: ~~E~~ ~~B~~ ~~D~~ ~~S~~ ~~A~~ ~~C~~ ~~F~~ ~~G~~ ~~H~~





task is: find the shortest path length from E to all other nodes

↳ # of edges

E – S – A – B 3

E – B 1 ← shortest path

DFS is not good for shortest paths; goes deep first

$O(|V| + |E|)$

BFS (G, u) queue (FIFO)

$d(u) = 0$ ← distance

$Q = [u]$

BFS while Q not empty

u = eject Q

for all ...

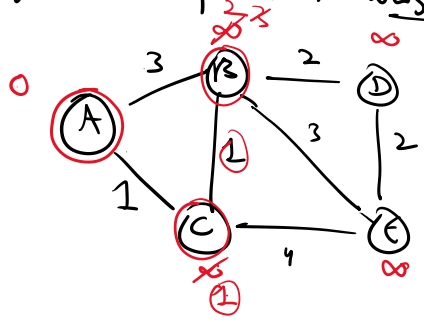
DFS (G, u): stack (FILO)

BFS

```
u = eject Q
for all v ∈ N(u):
    if not visited(v):
        add v to Q ; set visited flag
        d(v) = d(v) + 1
```

---

finding shortest paths in weighted graphs



$G = (V, E)$

$w((u,v))$ = weight on edge

weight function

$A \overset{3}{—} B \overset{3}{—} E = 6$

$A \overset{1}{—} C \overset{4}{—} E = 5$

keep the BFS frontier @ shortest distance.