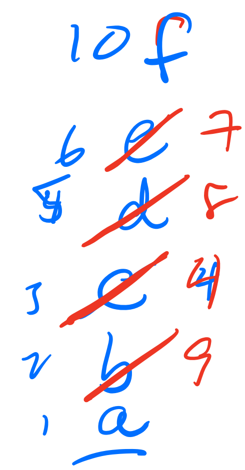
DFS (cont) undirected graphs Chapter 3

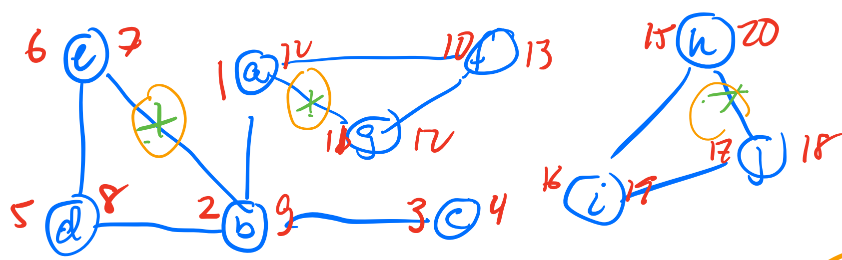
```
Procedure Explore(G, v)
    visited(v) = true
    previsit(v)
    for each edge (v,u) ∈ E
        if not visited(u):
            explore(G,u)
    postvisit(v)
```

previsit(v)
pre(v) = clock++

postvisit(v)
post(v) = clock++

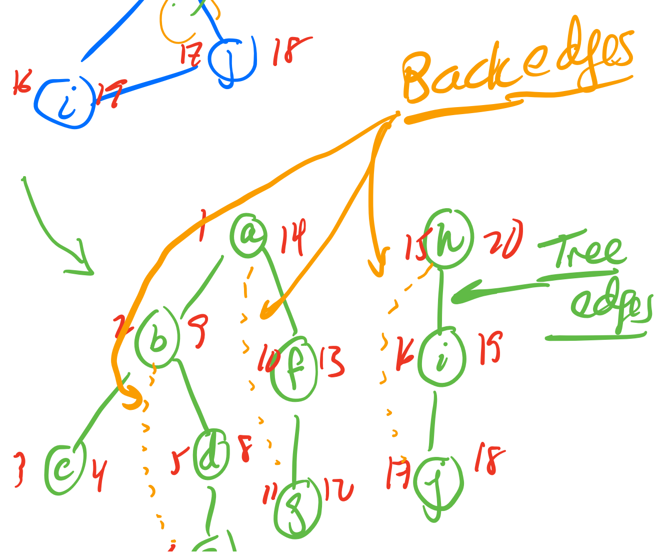
```
Procedure DFS(G)
    for all v ∈ V
        visited(v) = false

    for all v ∈ V:
        if not visited(v):
            explore(G,v)
```



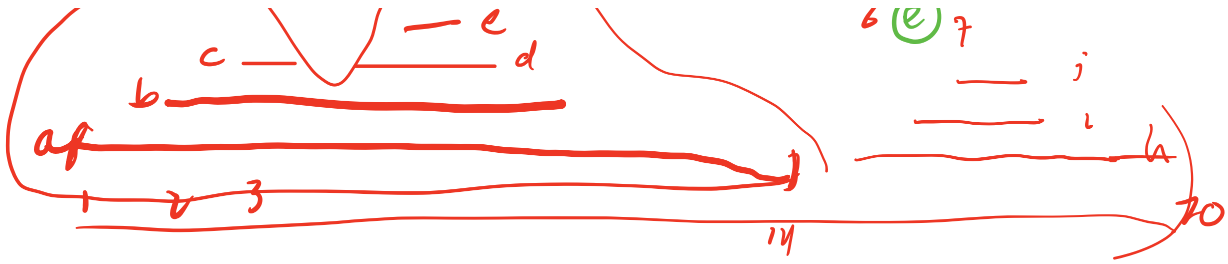


2 connected components

Draw numberline



DFS forest



Never have $pre(v) < pre(u) < post(v) < post(u)$

This represents a **stack**

How long does DFS(G) take?

- Explore is called once per vertex
  $O(n)$
- $O(1)$ + time for inner loop

$\therefore$ total time $O(V)$ + time inner loops

During inner loops, each edge is examined twice, once from each endpoint

$\therefore$ total time is $O(n+m)$

Wrapup: $[pre(u), post(u)]$ is the time

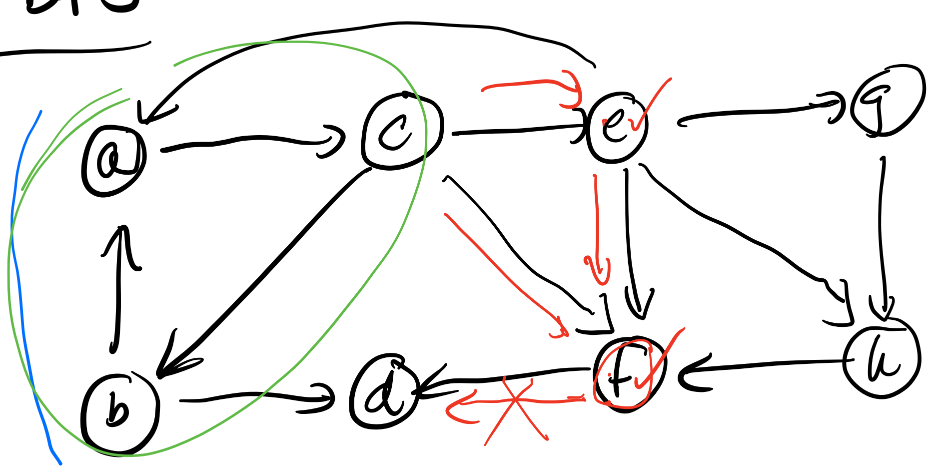
u is on the stack

# of connected components = # calls of explore from DFS when visited(u) is false.
explore(u) gives the whole component

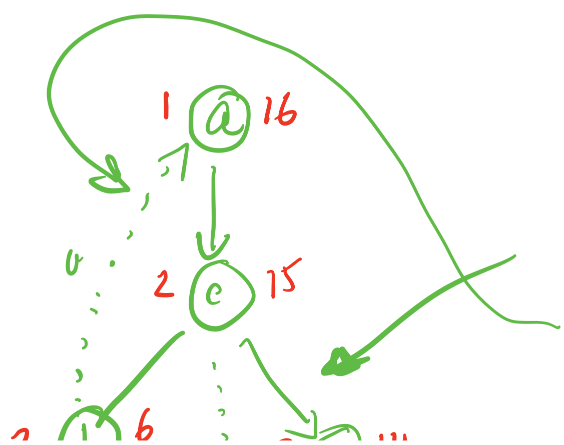
Directed DFS

explore(G,v)
DFS(G)
previsit(v)
postvisit(v)

SAME



Stack

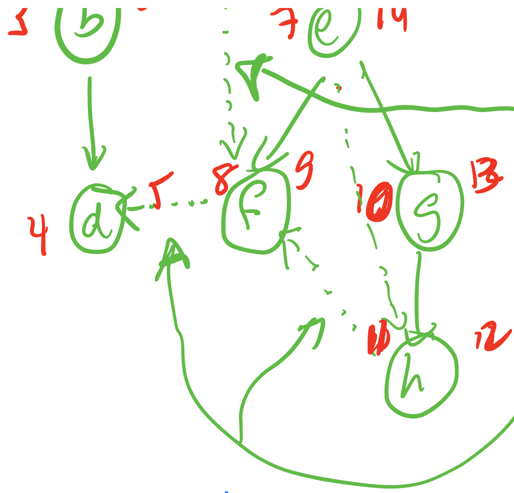


Four types of edges

tree edges

back edges

lead to an ancestor



forward edges
lead to a non-child descendant

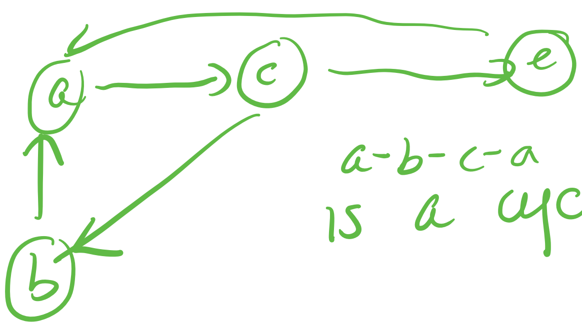
cross edges
everything else

Node u is an ancestor of v

Cycles A cycle is a circular path of directed edges

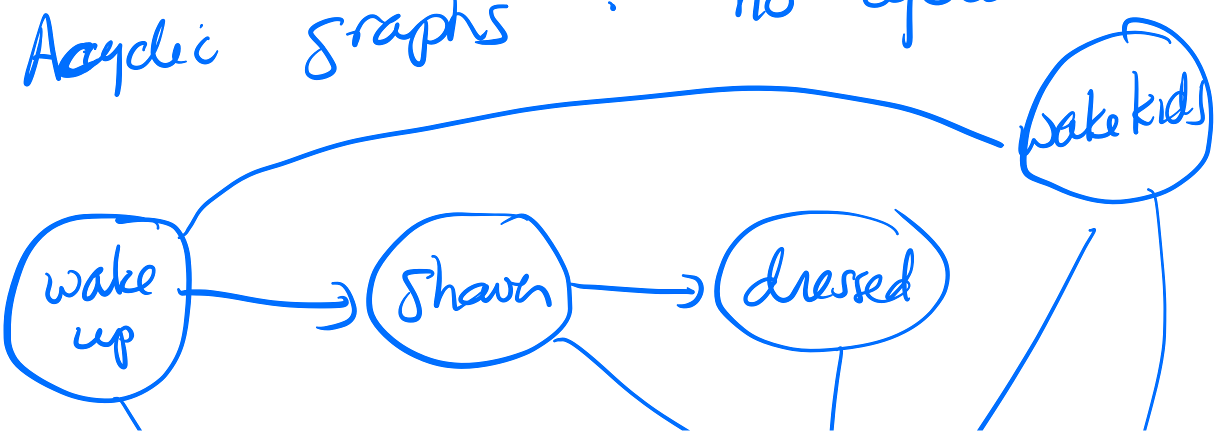
$v_0 \to v_1 \to v_2 \to \dots \to v_k \to v_0$

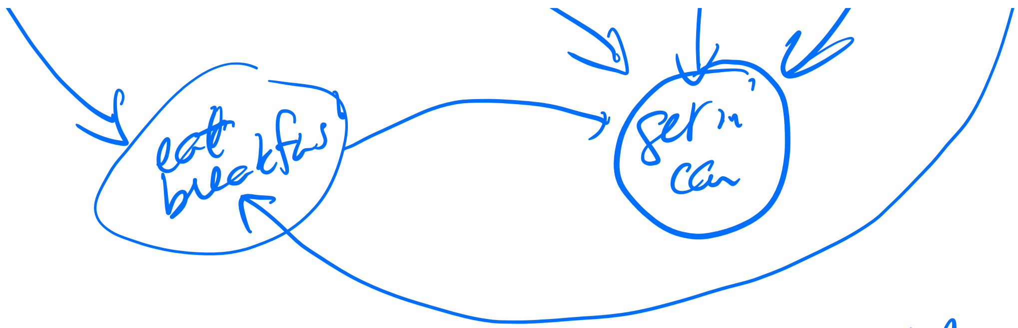
a-c-e-a is also a cycle



a-b-c-a is a cycle

Acyclic graphs : no cycles



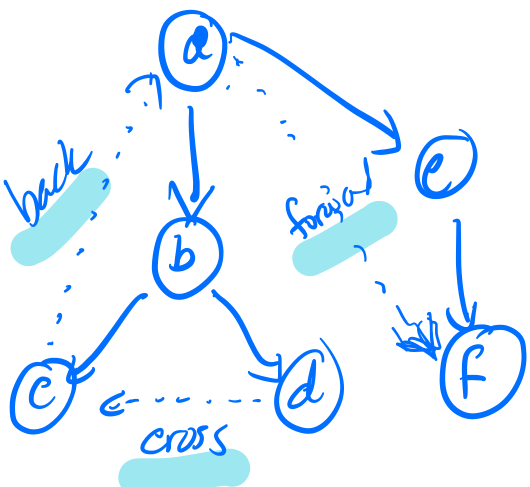


If you have a DAG — Directed Acyclic Graph

then there exists a topological sort

∩ linear ordering

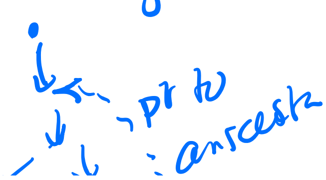
∩ topological ordering



Note:

Claim: A directed graph G has a cycle if and only if it has a back edge.

If we have a cycle →



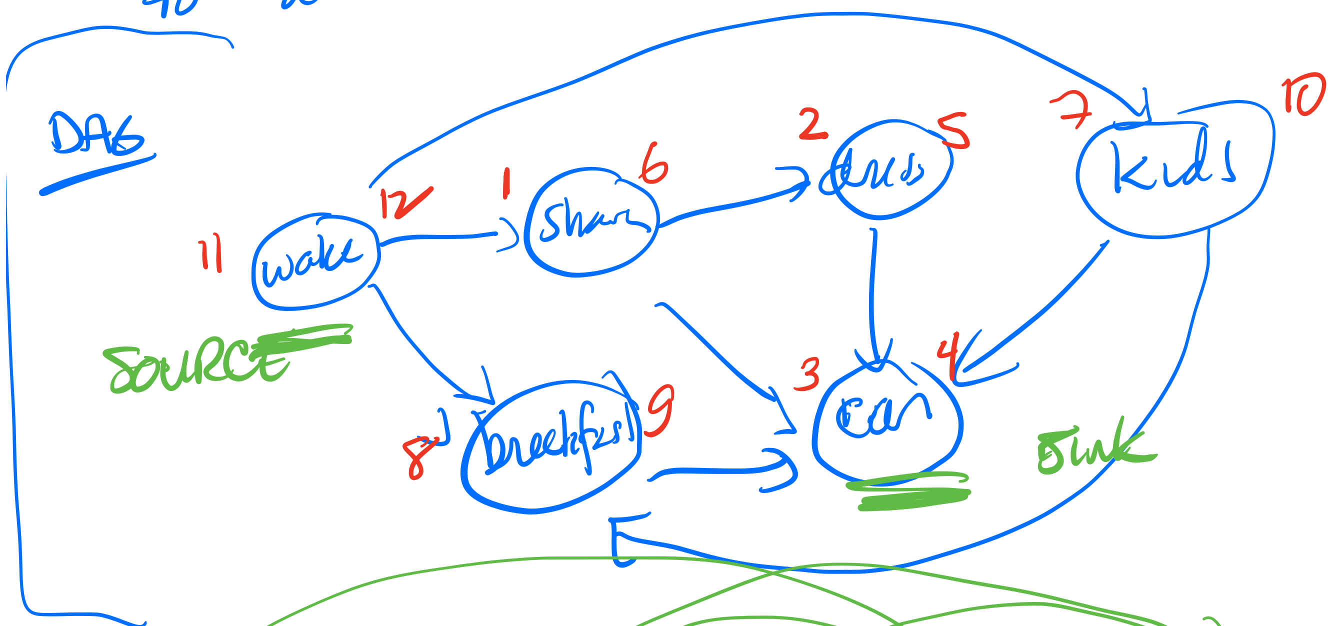
Back edge

Returning to a prev vertex on path from root → cycle backedge

Soln

Run DFS & perform tasks in order of decreasing post numbers.

Claim: In a DAG every edge leads to a lower post number



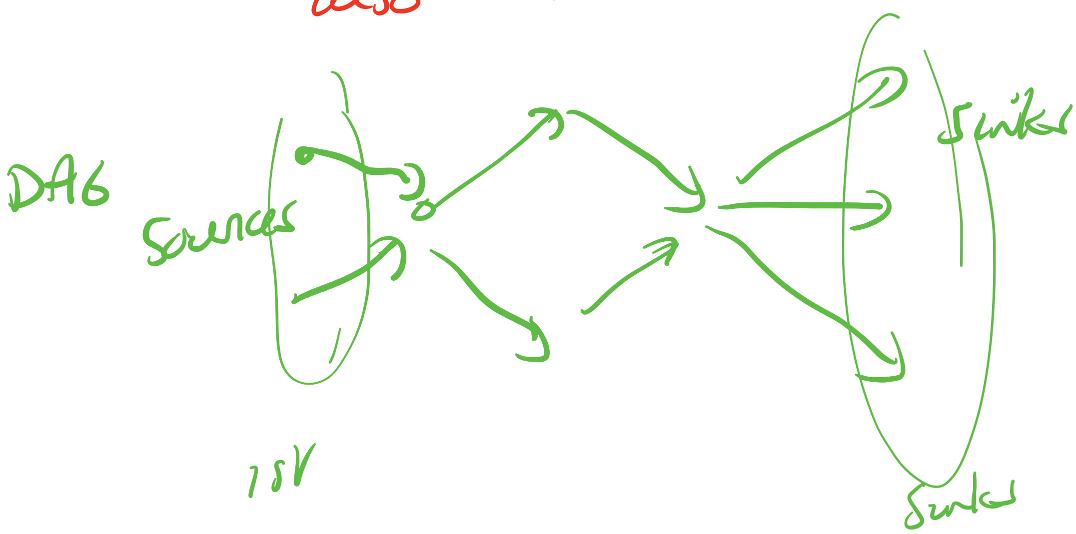
wake, → shower, → kids, → dress, → breakfast, → car

topological sort: ~~total~~ linear ordering

if all graph edges left to right

wake → kids → khan → dress → breakfast → car

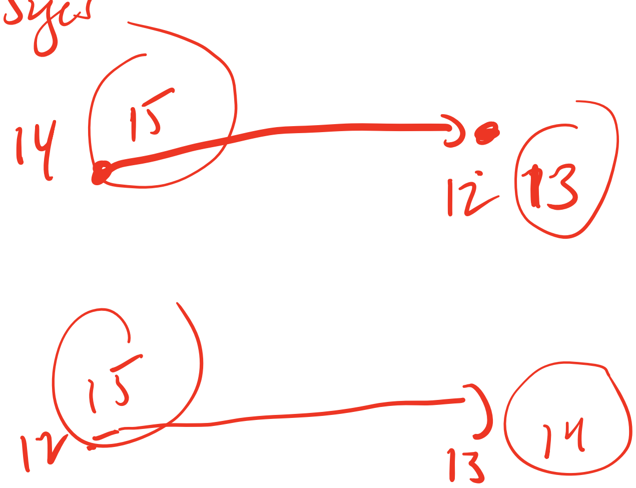
also valid



There are sinks because keep going forwards + you have to stop

There are sources because reverse edges + look for sinks in reverse graph!

Relative Sizes



In both cases post # decreases

output vertices in descending post #

= linear ordering