Algorithms - Spring '25

DFS, BFS,
& variants

Recap

- Oral grading starts today
  (No office hours tomorrow)

- Exam next Tuesday
  - review session Monday
  - practice exam posted

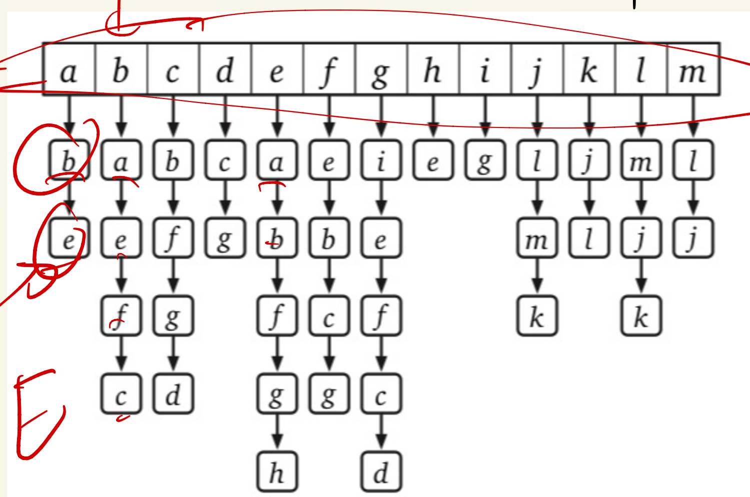
- Reading due Friday, then
  <u>next</u> Wednesday

Graph Searching

How can we tell if 2 vertices are connected?

Remember, the computer only has:

V



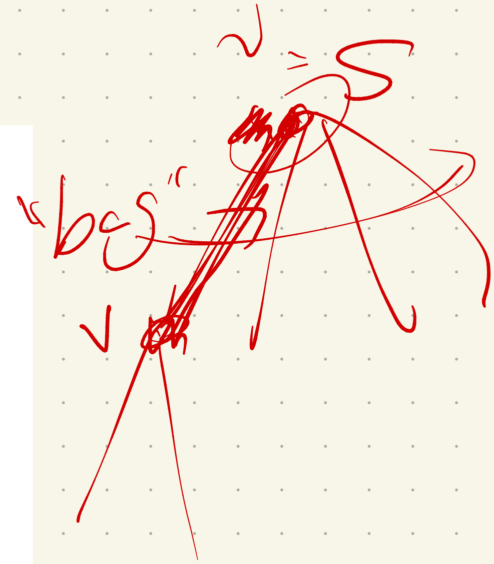
$\{u,v\} \in E$

is a connected to m?

Bigger question: can we tell if all the vertices are in a single connected component?

Possibly you saw depth first search (DFS) and breadth first search (BFS) in data structures:

```
WhateverFirstSearch(s):
    put s into the bag
    while the bag is not empty
        take v from the bag
        if v is unmarked
            mark v
            for each edge vw
                put w into the bag
```



These are essentially just search strategies!

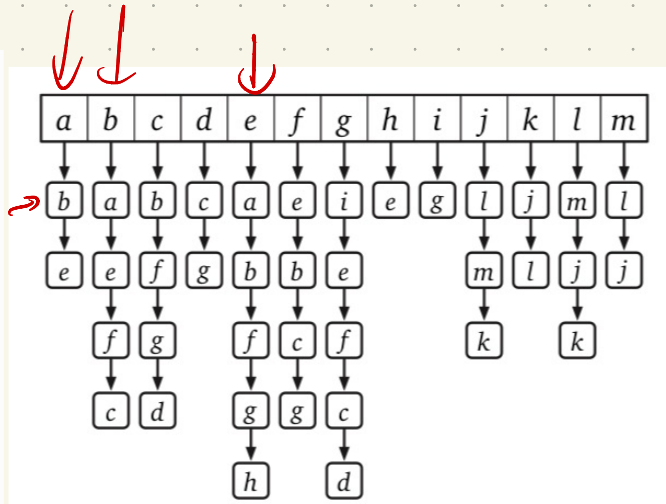
How can we decide if u + v are connected?

Q: what "bag"?

lots of data structures!

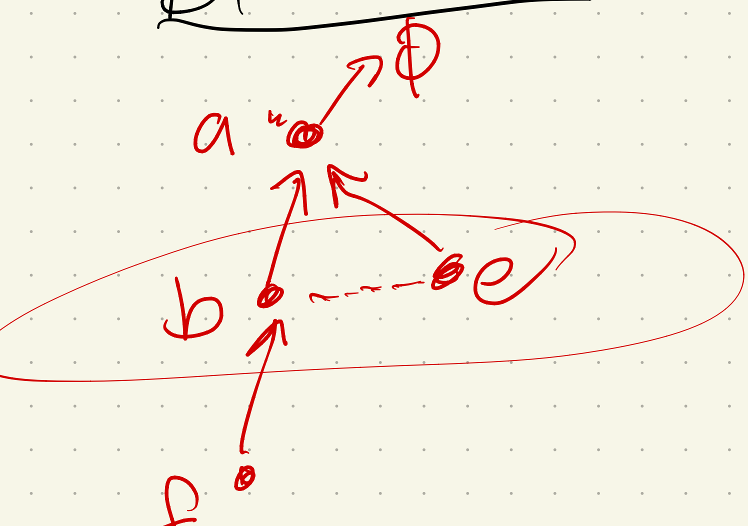
Can use this to build a <u>spanning tree</u>:

WhateverFirstSearch(s):
put $(\varnothing, s)$ in ~~bag~~ queue
while the bag is not empty
  take (p, v) from the bag $(\star)$
  if v is unmarked
    mark v
    $parent(v) \leftarrow p$
    for each edge vw $(\dagger)$
      put (v, w) into the bag $(\star\star)$



"~~bag~~": $\Theta(1)$ per operation

<u>BFS tree</u>:



queue:

front ~~(Φ, a)~~, ~~(a, b)~~, back

~~(a, e)~~, (I) add

remove

~~(b, a)~~ ~~(b, e)~~,

(b, f), (b, c),

(e, a), (e, b), (e, f)... ~~esbns~~

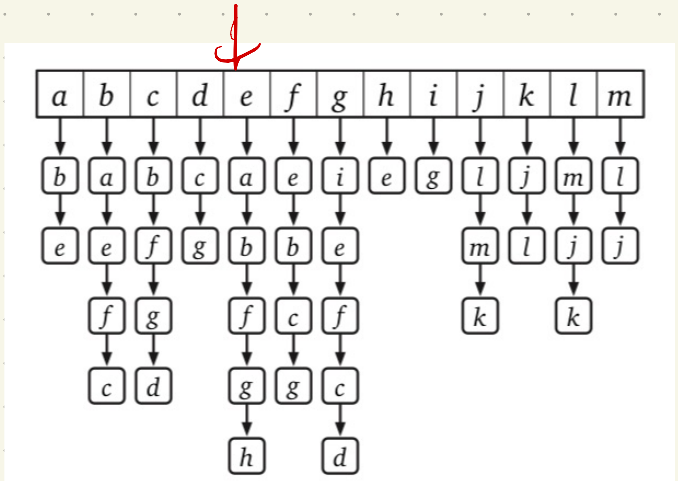
$(p, v) =$ ~~(Φ, a)~~

~~(a, b)~~ ~~(b, e)~~ (b, f)

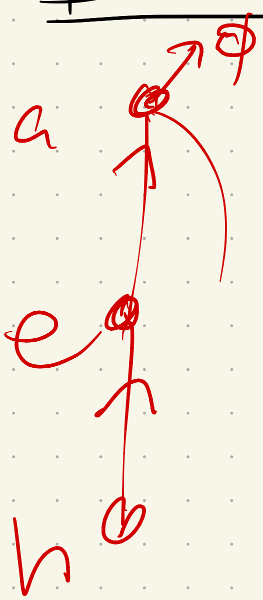
~~(a, e)~~

~~(b, a)~~

```
WHATEVERFIRSTSEARCH(s):
  put (∅, s) in bag
  while the bag is not empty
      take (p, v) from the bag                    (⋆)
      if v is unmarked
          mark v
          parent(v) ← p
          for each edge vw                        (†)
              put (v, w) into the bag             (⋆⋆)
```



DFS tree



stack: O(1)

h's nbrs

~~(e,h)~~

(e,g)

(e,f)

(e,b)

(e,a)

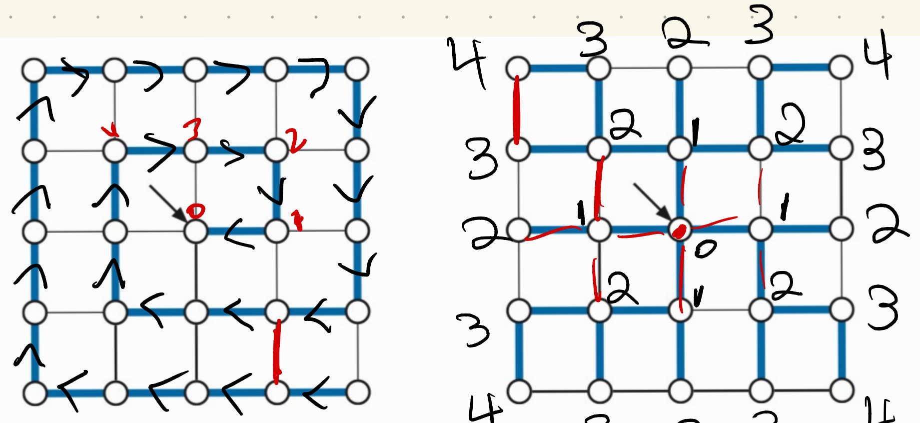
~~(a,e)~~

(a,b)

~~(∅,a)~~

stack:

Just remember: different!

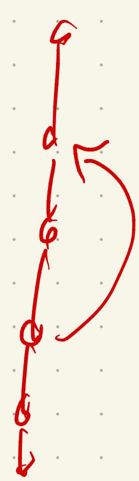


levels

Figure 5.12. A depth-first spanning tree and a breadth-first spanning tree of the same graph, both starting at the center vertex.

DFS:

All non-tree edges must connect a vertex to an ancestor in the tree



BFS:

All non-tree edges must connect vertices either at the same level, or 1 level apart

Runtime:

```
WHATEVERFIRSTSEARCH(s):
    put s into the bag                ← O(1)
    while the bag is not empty        ← O(1)
        take v from the bag           ← O(1)
        if v is unmarked              ← O(1)
            mark v
            for each edge vw
                put w into the bag
```

Think of each edge:
only put on the
stack/queue ≤ 2
time

each edges costs
O(1) over lifetime of
alg.

If we have connected, O(V+E)

Correctness:

Claim: WFS will mark all reachable vertices.

Pf: induction on distance to the source:

d=0: Then vertex = source! We know this is marked at beginning - see first lines!

d>0: Consider v at distance d, so $S \to v_1 \to v_2 \cdots \to v_{d-1} \to v$ in G.

$\underbrace{\qquad\qquad\qquad}_{d \text{ edges}}$

assume any vertex at dist < d is marked

By IH: know v_{d-1} is marked

That means v_{d-1} was marked!

```
WHATEVERFIRSTSEARCH(s):
    put s into the bag
    while the bag is not empty
        take v from the bag
        if v is unmarked
            mark v
            for each edge vw
                put w into the bag
```

started unmarked, so this line of code ran with $v = v_{d-1}$

At this point, edge $\{v_{d-1}, v_d\}$ is added to bag.

When alg terminates, v_d will have been popped + marked. $\boxed{\text{///}}$

Claim: marked v's + parents form a spanning tree.
(See demo's...) ← (past later)

proof:

```
WHATEVERFIRSTSEARCH(s):
  put (∅, s) in bag
  while the bag is not empty
      take (p, v) from the bag                  (⋆)
      if v is unmarked
          mark v
          parent(v) ← p
          for each edge vw                      (†)
              put (v, w) into the bag           (⋆⋆)
```

For each marked vertex:

marked once at which point (v, p) is added to tree

(except s! because (s, ∅) is in tree)

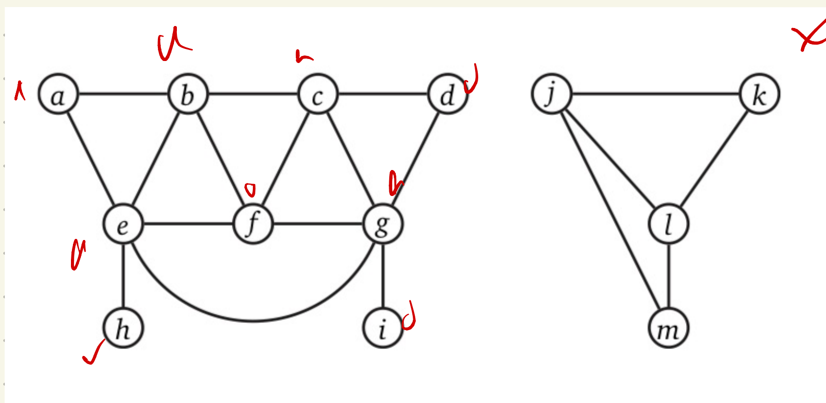
n vertices, n-1 edges, connected ⟹ tree. ∎

In a disconnected graph:

Often want to count or label the components of the graph!

(WFS(v) will only visit the piece that v belongs to.)

not marked



Solution: Call it more than one time!

unmark all vertices
for all vertices v:
call WFS(v)
while ~~if~~ any vertex w is unmarked
WFS(w)

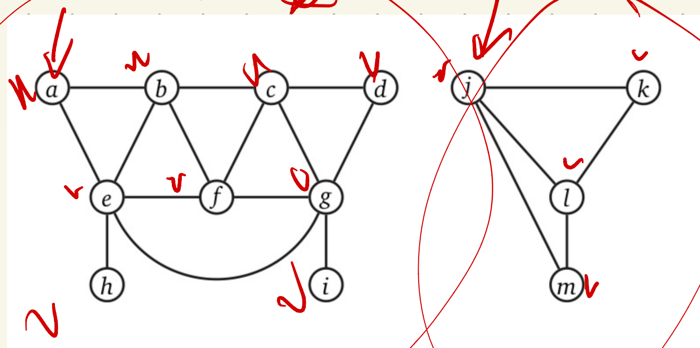
Modification: Might want to count the # of connected components?

```
CountComponents(G):
    count ← 0
    for all vertices v
        unmark v
    for all vertices v
        if v is unmarked
            count ← count + 1
            WhateverFirstSearch(v)
    return count
```

$O(V)$

across alg. $O(V+E)$

V_1+E_1



V_2+E_2

Count = 2

$O(c \cdot V+E)$

$\Rightarrow (\#\text{comps}) \cdot V + O(V+E)$

$\Rightarrow \leq V^2+E$

Finally, can even record which component each vertex belongs to:

```
COUNTANDLABEL(G):
  count ← 0
  for all vertices v
      unmark v
  for all vertices v
      if v is unmarked
          count ← count + 1
          LABELONE(v, count)
  return count
```

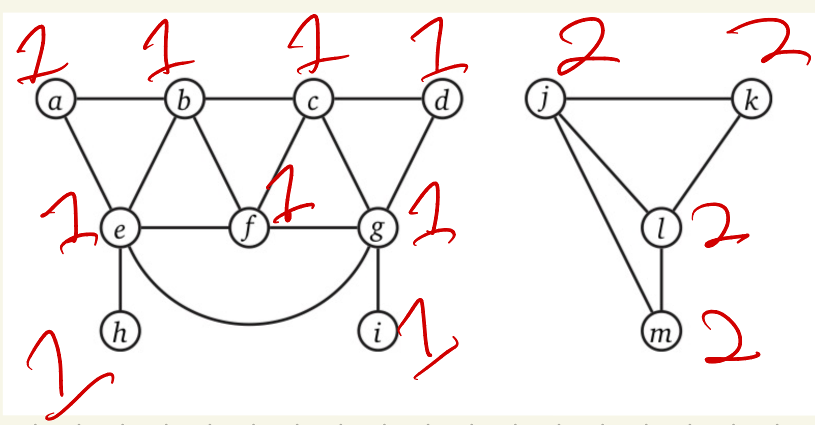
WTS

```
⟨⟨Label one component⟩⟩
LABELONE(v, count):
  while the bag is not empty
      take v from the bag
      if v is unmarked
          mark v
          comp(v) ← count
          for each edge vw
              put w into the bag
```

marked: T/F for each vertex

comp[1..V]

array



O(1): comp[u] = comp[v] ?

<u>Dfn</u>: Reduction

A <u>reduction</u> is a method of solving a problem by transforming it to another problem.

<u>Note</u>: you've seen/done this in other classes!

We'll see a ton of these!

(Especially common in graphs...)

<u>Key</u>: describe how to build a graph

First example:

Given a pixel map, the flood-fill operation lets you select a pixel & change the color of it & all the pixels in its region.

nxn grid, + value in each cell

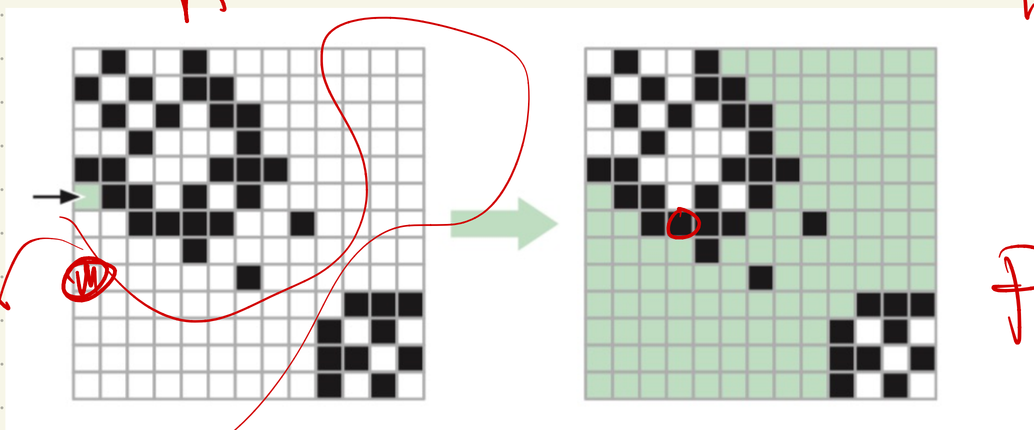


Figure 5.13. An example of flood fill

P[n][n]

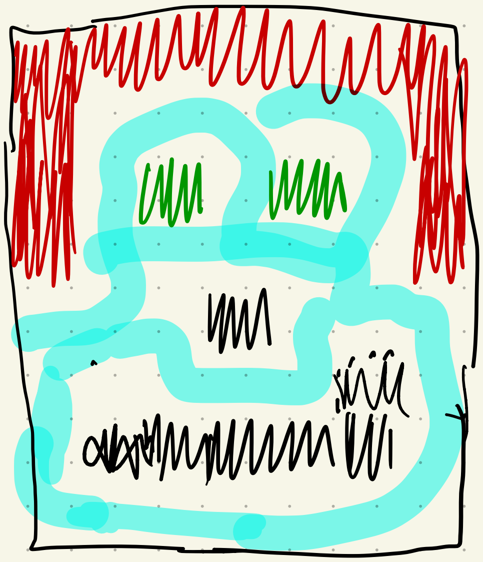
How?

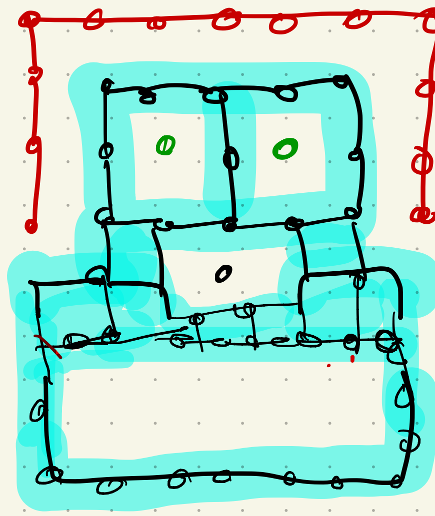
Convert to a graph problem, & call WFS

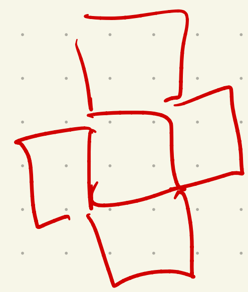
BFS or DFS

look at marked vertices

So: Build a graph from pixels:







Build graph: $V = n^2$ $E \leq 4n^2$ Make one vertex per cell.

Add 1-4 nbrs as edge

Algorithm

If pixel p is selected:

call WFS (p's vertex)

reset color on any marked vertex's pixel

Runtime: in terms of input!

$\Rightarrow O(n^2)$

Arguably, these reductions are the most important thing in graphs!

Like data structures - you won't usually have to re-code everything.

Instead:

- Set up graph : $O(n^2)$
- Call some algorithm : $O(n^2)$

$O(V+E) = O(n^2 + 4n^2)$
$= O(n^2)$

So runtime/correctness:

Built correct graph so that alg finds right object

Next chapter:

All about directed graphs!

First, though, some things to recall: graph traversals.

- Pre-order: visit v
  visit children

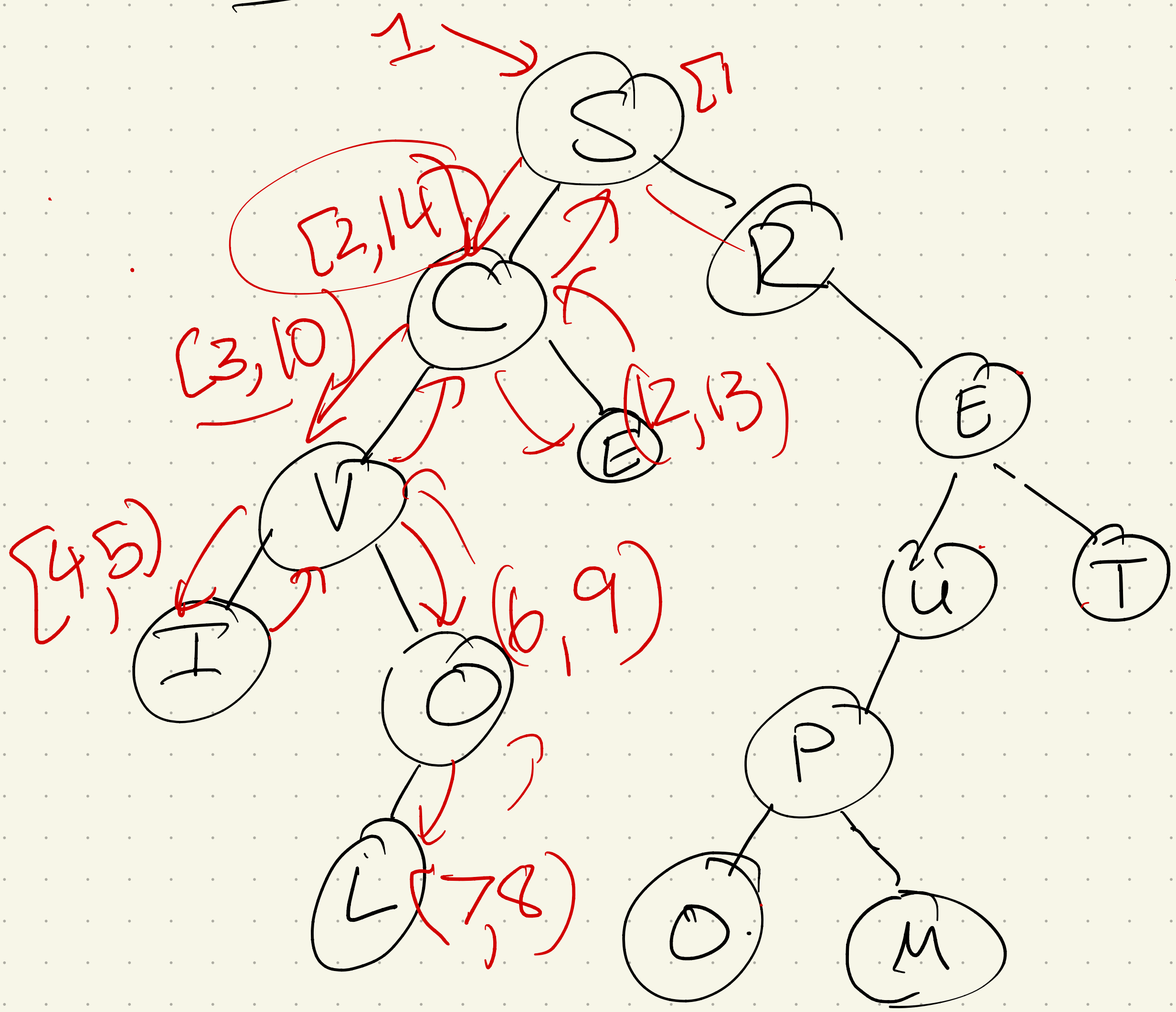
- Post-order: visit all children
  visit v

- In-order: (binary tree)

left
self
right

Searching & directed graphs:

Recall: post order traversal



- imagine a "clock" incrementing each time an edge is traversed:

Result:

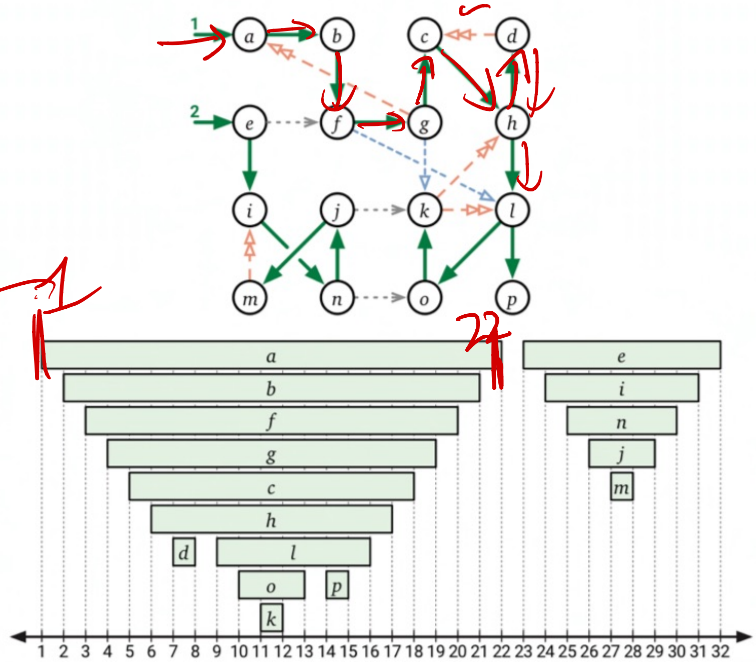


Figure 6.4. A depth-first forest of a directed graph, and the corresponding active intervals of its vertices, defining the preordering *abfgchdlokpeinjm* and the postordering *dkoplhcgfbamjnie*. Forest edges are solid; dashed edges are explained in Figure 6.5.

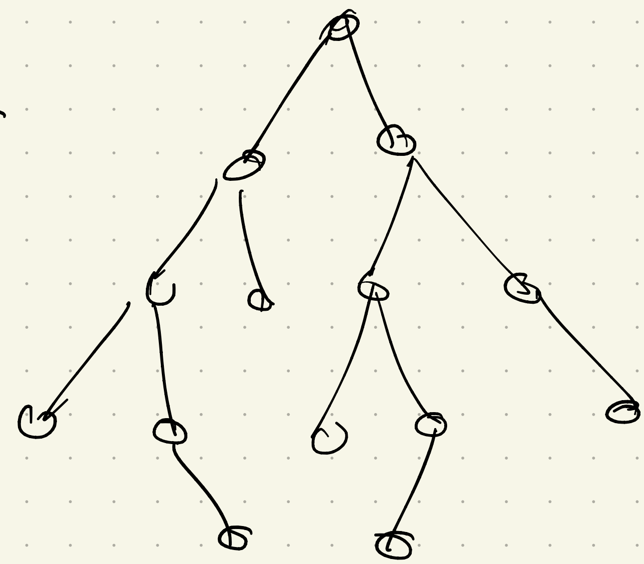
So: in DFS, this "lifespan" represents how long a vertex is on the stack.

Notation:

[v.pre, v.post]

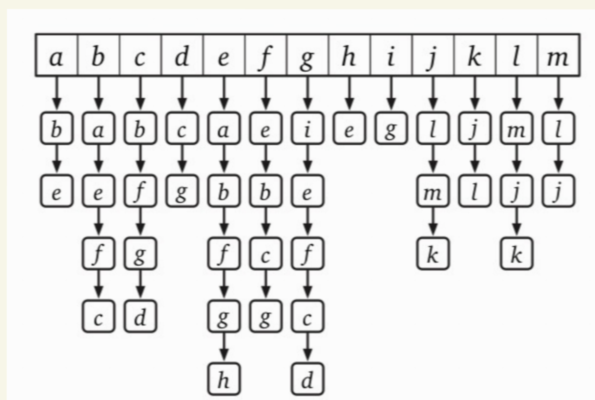
Note: In general graphs, post order traversal is **not** unique!

It was in BSTs:



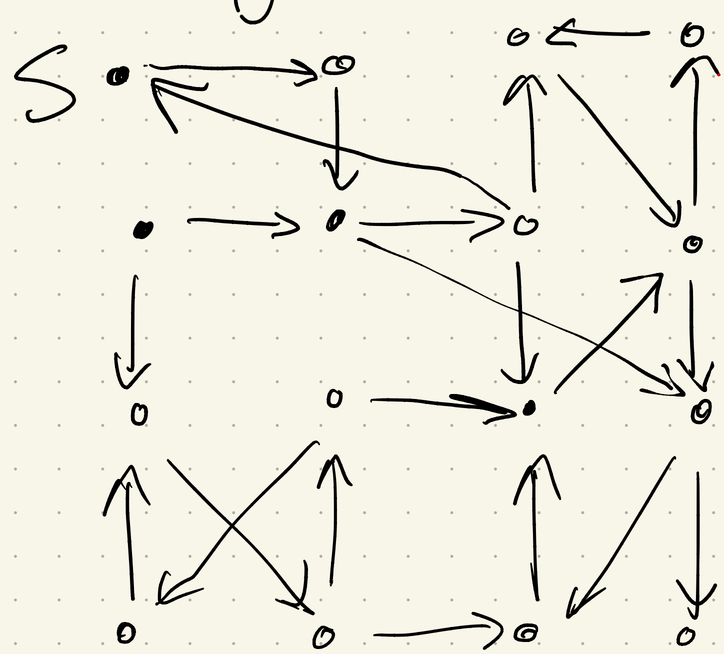
In graphs:
Just use adj. list order.

```
DFS(v):
  if v is unmarked
    mark v
    for each edge v→w
      DFS(w)
```



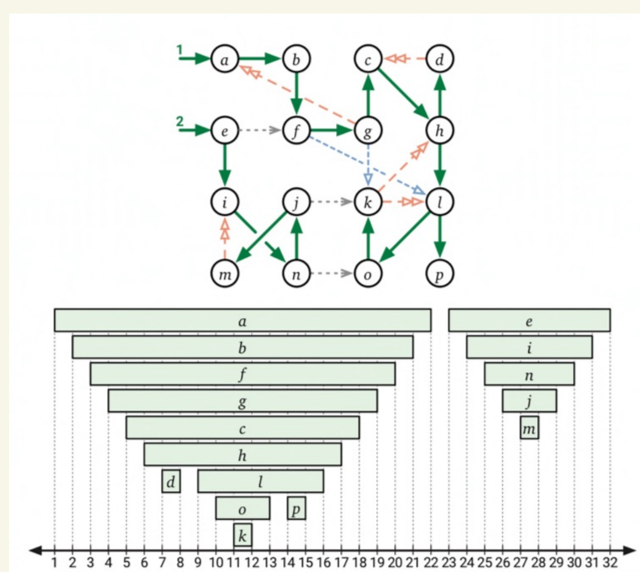
Dfn: - tree edge
- forward edge
- back edge
- cross edge

Picture:



DFS tree

Clock reference:



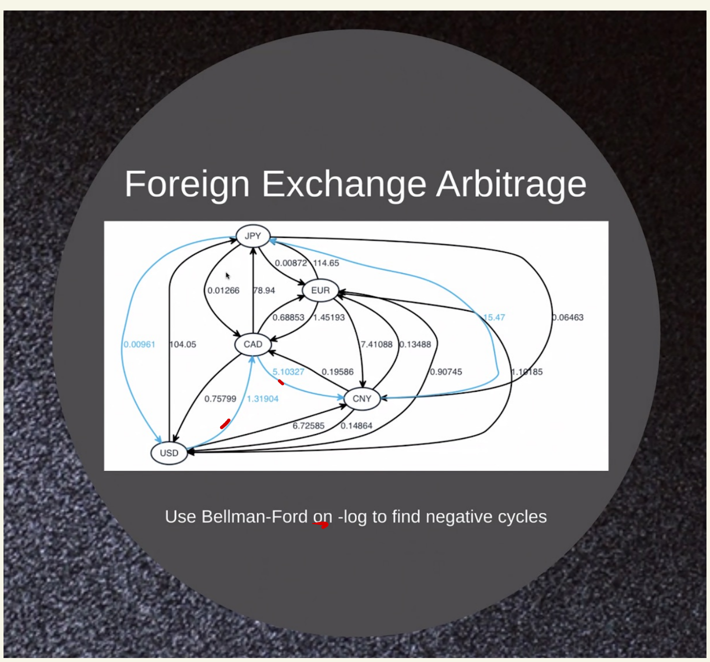
Finding cycles

In general, cycles tend to be important.

Sometimes bad:

- topological ordering in a DAG (see next slides)
- longer run time
  ↳ see Dyn. Pro.

Sometimes good:



(Taken from a talk on high freq. trading)