Algorithms - Spring '25

DFS, BFS, & variants

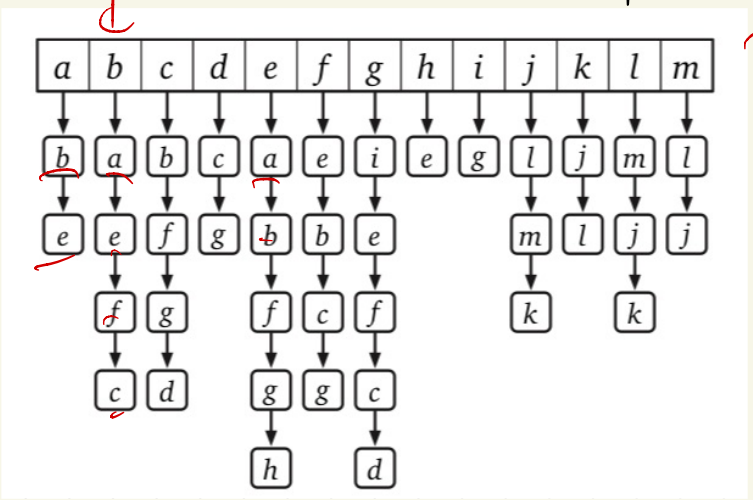
Recap

- Oral grading - starts today
  (No office hours tomorrow)
- Exam next Tuesday
  - review session Monday
  - practice exam posted
- Reading due Friday, then next Wednesday

Graph Searching

How can we tell if 2 vertices are connected?

Remember, the computer only has:

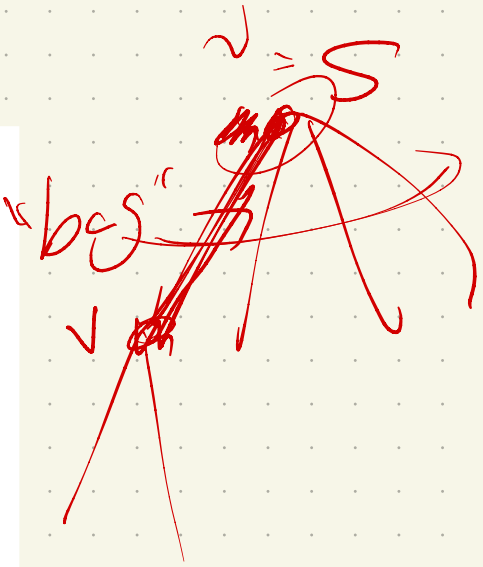


Is g connected to m?

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!

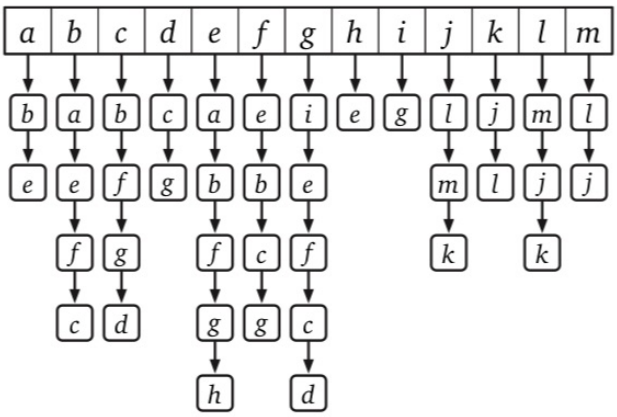
How can we decide if u & v are connected?

Q: what "bag"?

lots of data structures!

Can use this to build a spanning tree:

```
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         (⋆⋆)
```

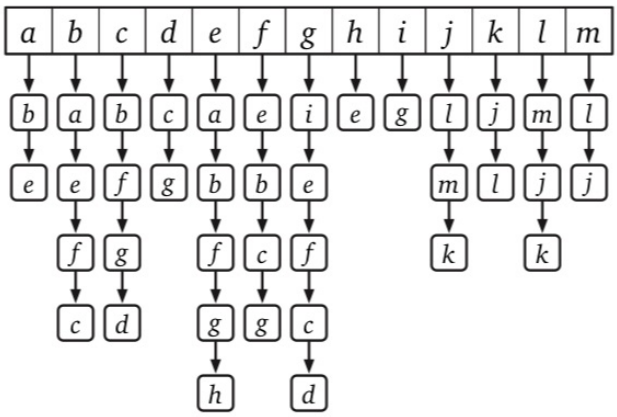


BFS tree:

queue:

front (∅, 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

(ϕ, a)

stack:

Just remember: different!

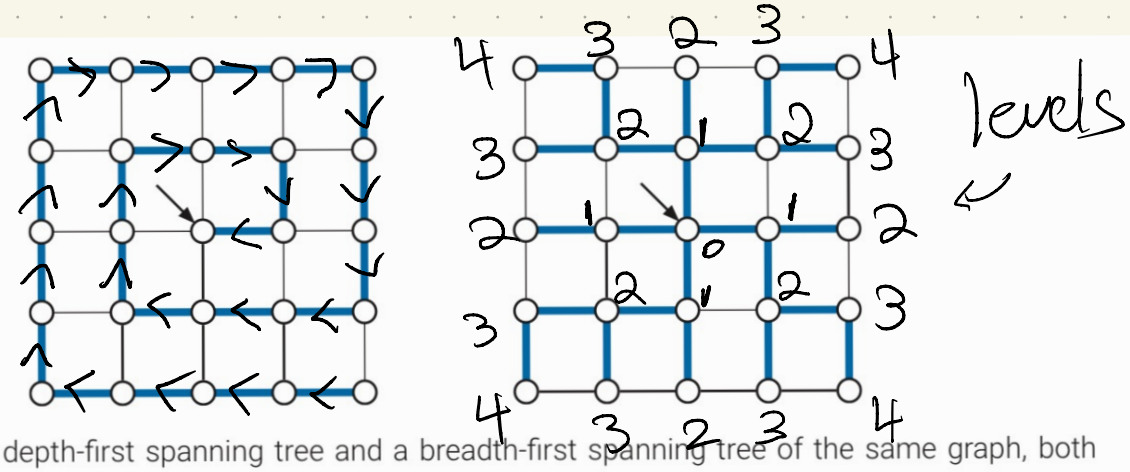


Figure 5.12. A depth-first spanning tree and a breadth-first spanning tree of the same graph, both starting at the center vertex.

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
  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
```

Correctness:

Claim: BFS will mark all reachable vertices.

Pf: induction on distance to the source:

$d=0$:

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

By IH:

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
```

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

proof:

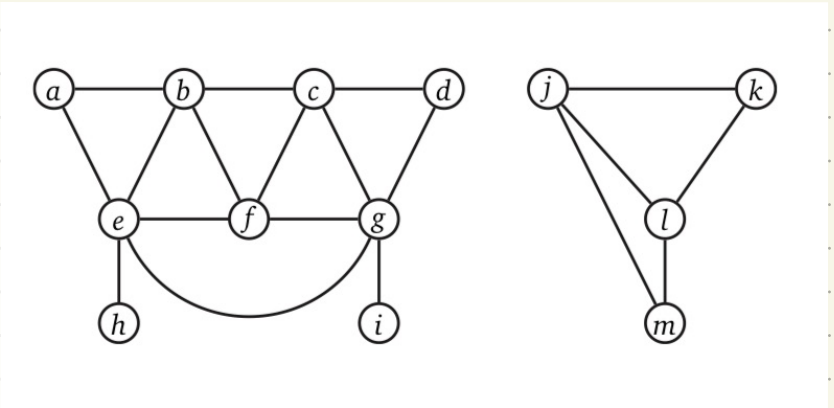
```
WHATEVERFIRSTSEARCH(s):
  put (∅, s) in bag
  while the bag is not empty
      take (p, v) from the bag                (⋆)
      if v is unmarked
          mark v
          parent(v) ← p
          for each edge vw                    (†)
              put (v, w) into the bag         (⋆⋆)
```

For each marked vertex:

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.)



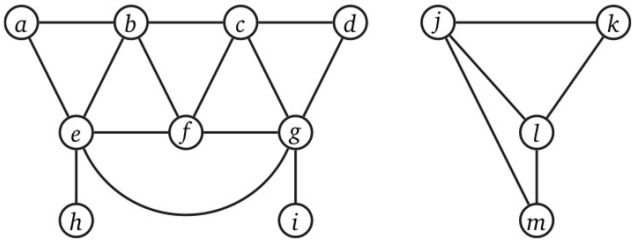
Solution: Call it more than one time!

unmark all vertices

For all vertices v:

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

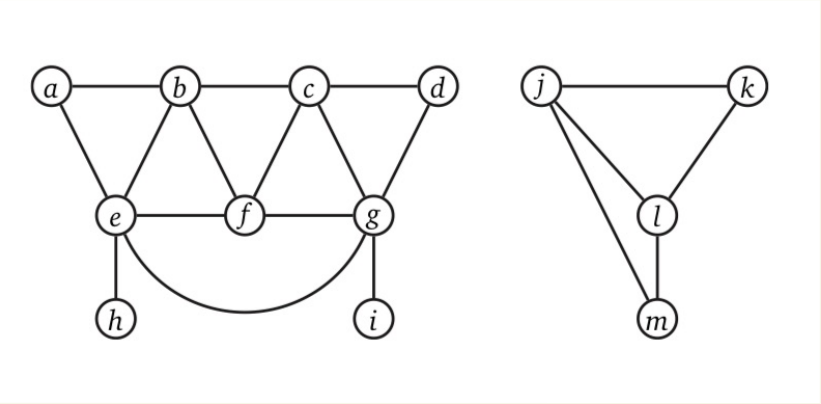
```
COUNTCOMPONENTS(G):
  count ← 0
  for all vertices v
      unmark v
  for all vertices v
      if v is unmarked
          count ← count + 1
          WHATEVERFIRSTSEARCH(v)
  return count
```



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
```

```
⟨⟨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
```



Dfn: Reduction

A reduction is a method of solving a problem by transforming it to another problem.

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

We'll see a ton of these!

(Especially common in graphs...)

Key:

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.

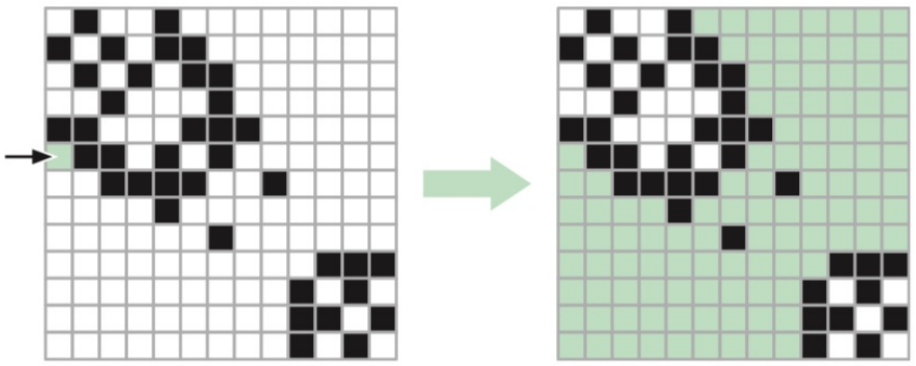
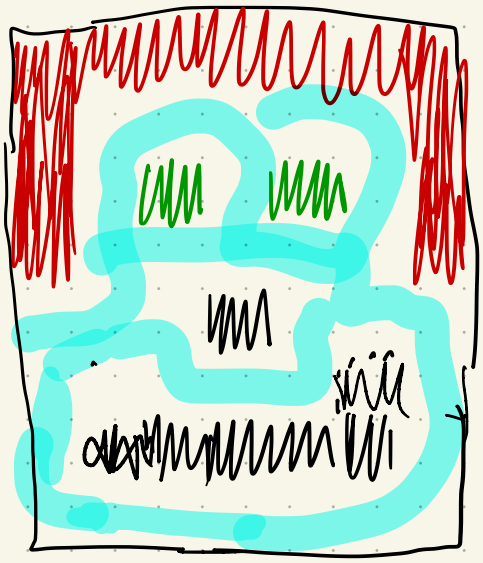
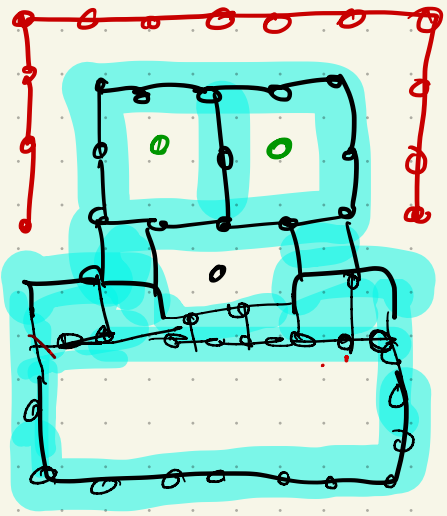


Figure 5.13. An example of flood fill

How?

So: Build a graph from pixels:





Algorithm

If pixel p is selected:

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
- Call some algorithm

So runtime/correctness:

Next chapter:

All about directed graphs!

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

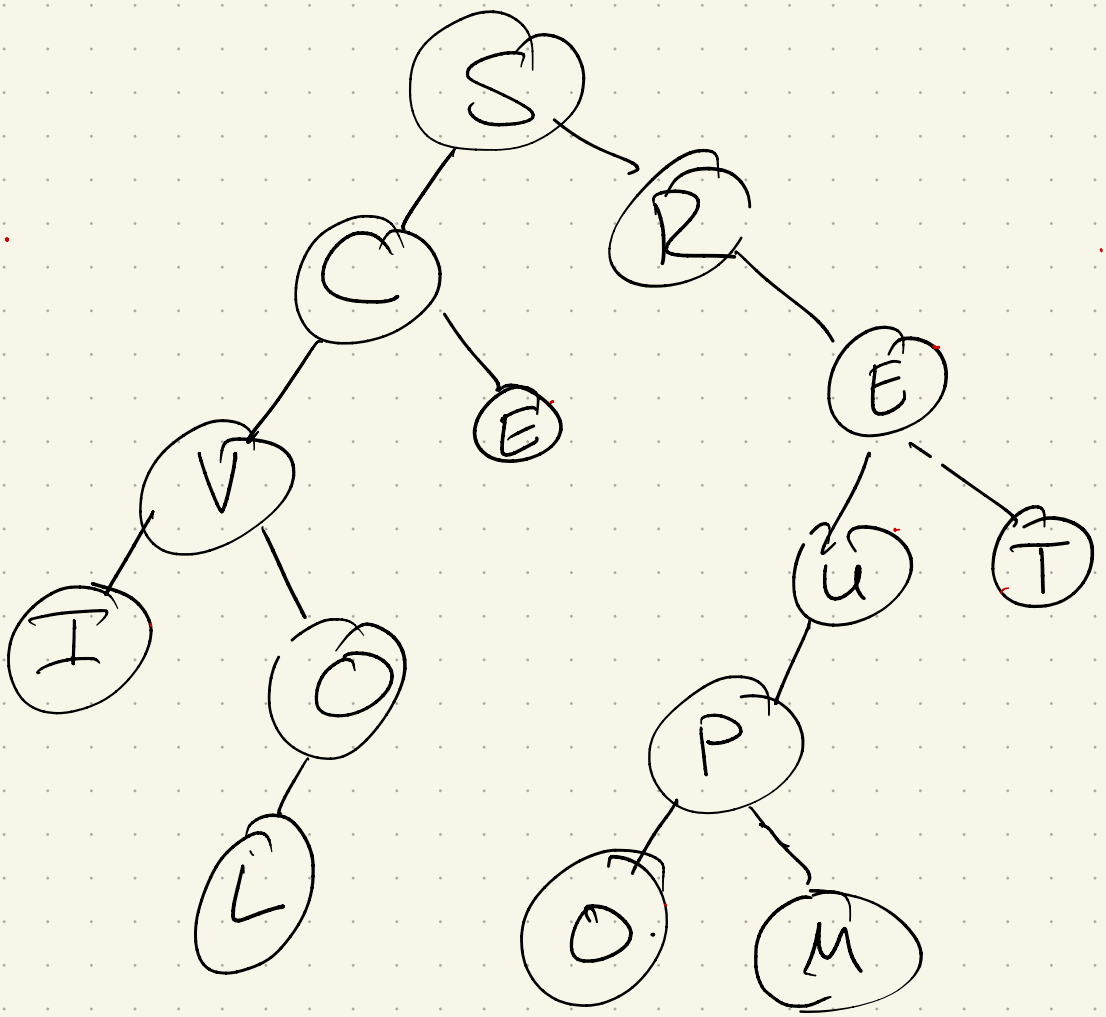
- Pre-order

- Post-order:

- In-order:

Searching a directed graphs:

Recall: post order traversal



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

Result:

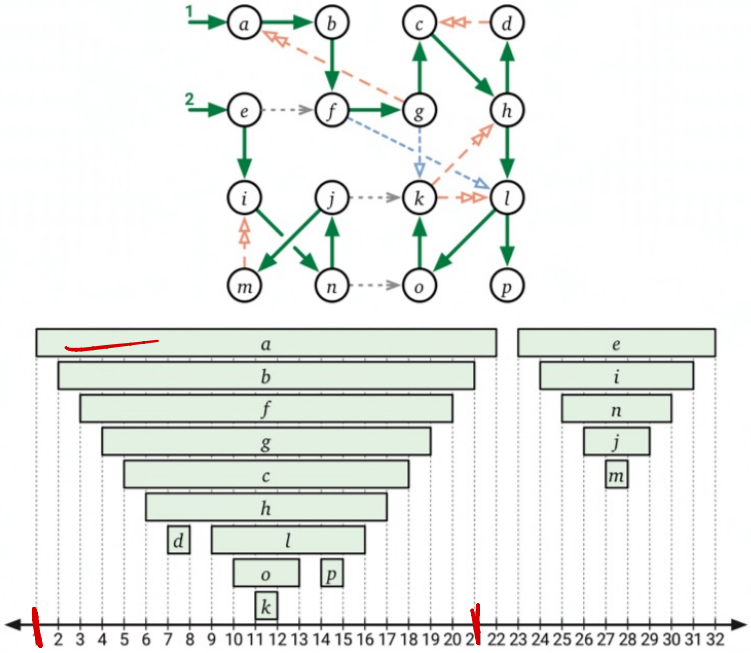


Figure 6.4. A depth-first forest of a directed graph, and the corresponding active intervals of its vertices, defining the preordering *abfgchdlokpeinjm* and the postordering *dkoplhcgfbamjnie*. Forest edges are solid; dashed edges are explained in Figure 6.5.

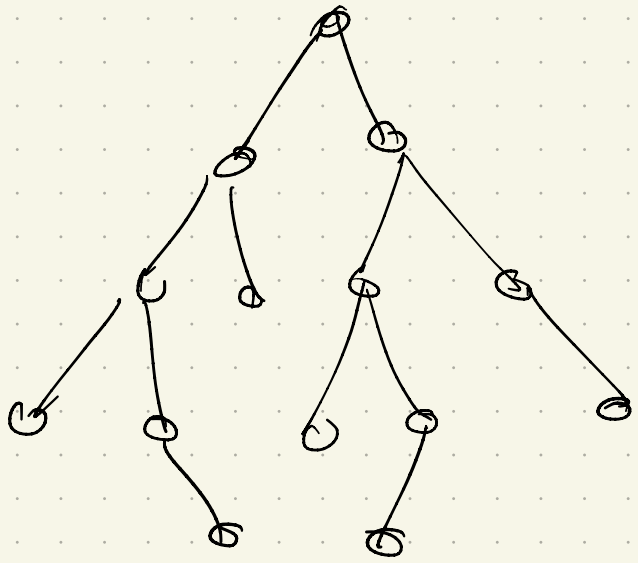
So: in DFS, this "lifespan" represents how long a vertex is on the stack.

Notation:

[v.pre, v.post]

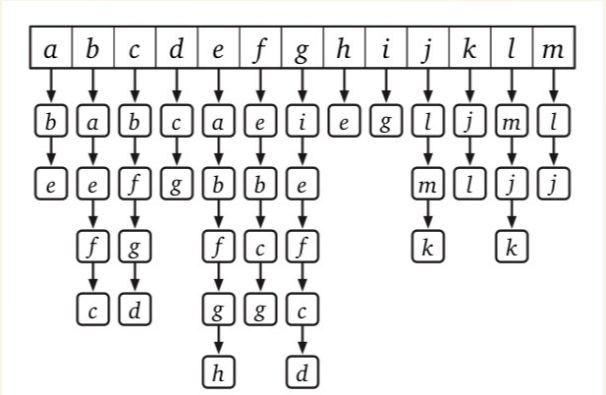
Note: In general graphs, post order traversal is not unique!

It was in BSTs:



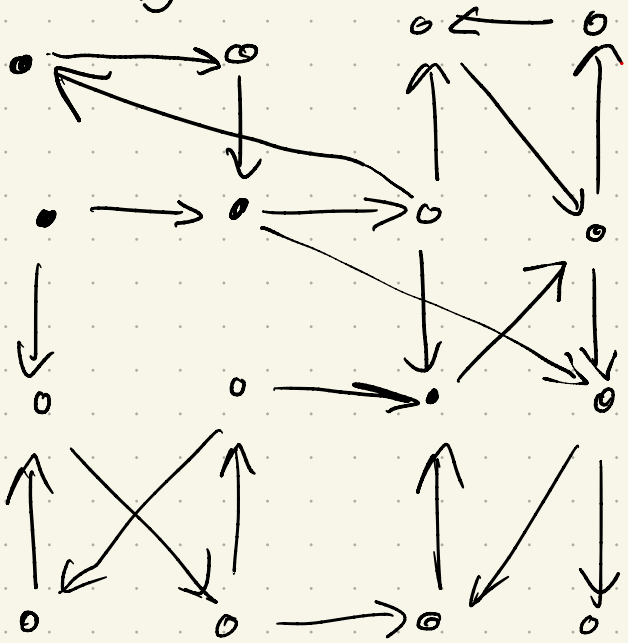
In graphs:
Just use adj. list order.

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



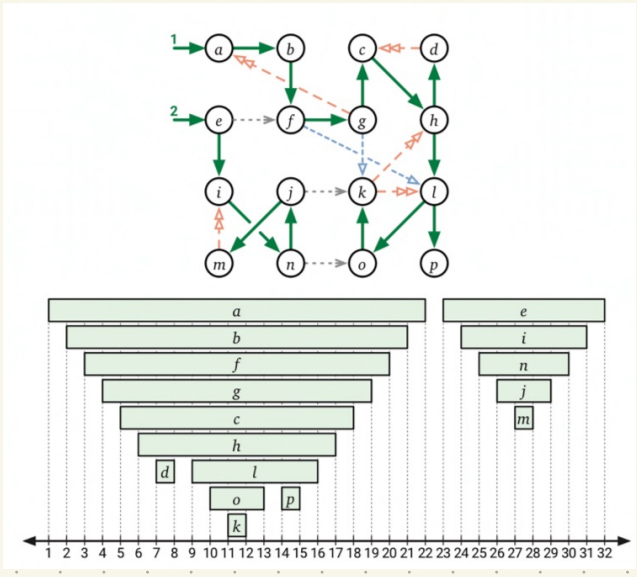
Dfn :- tree edge
- forward edge
- back edge
- cross edge

Picture : S



DFS tree

Clock reference :



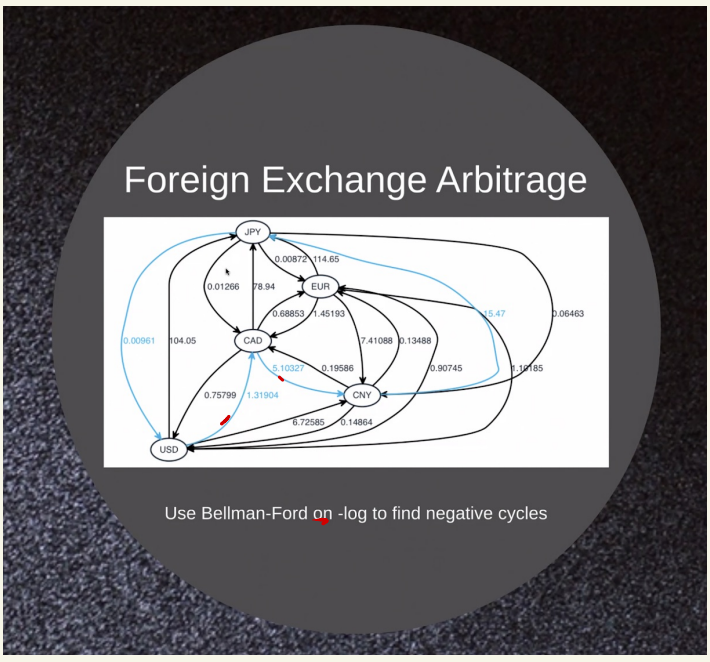
Finding cycles

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 colloquium last year, by a person who works in high frequency trading)