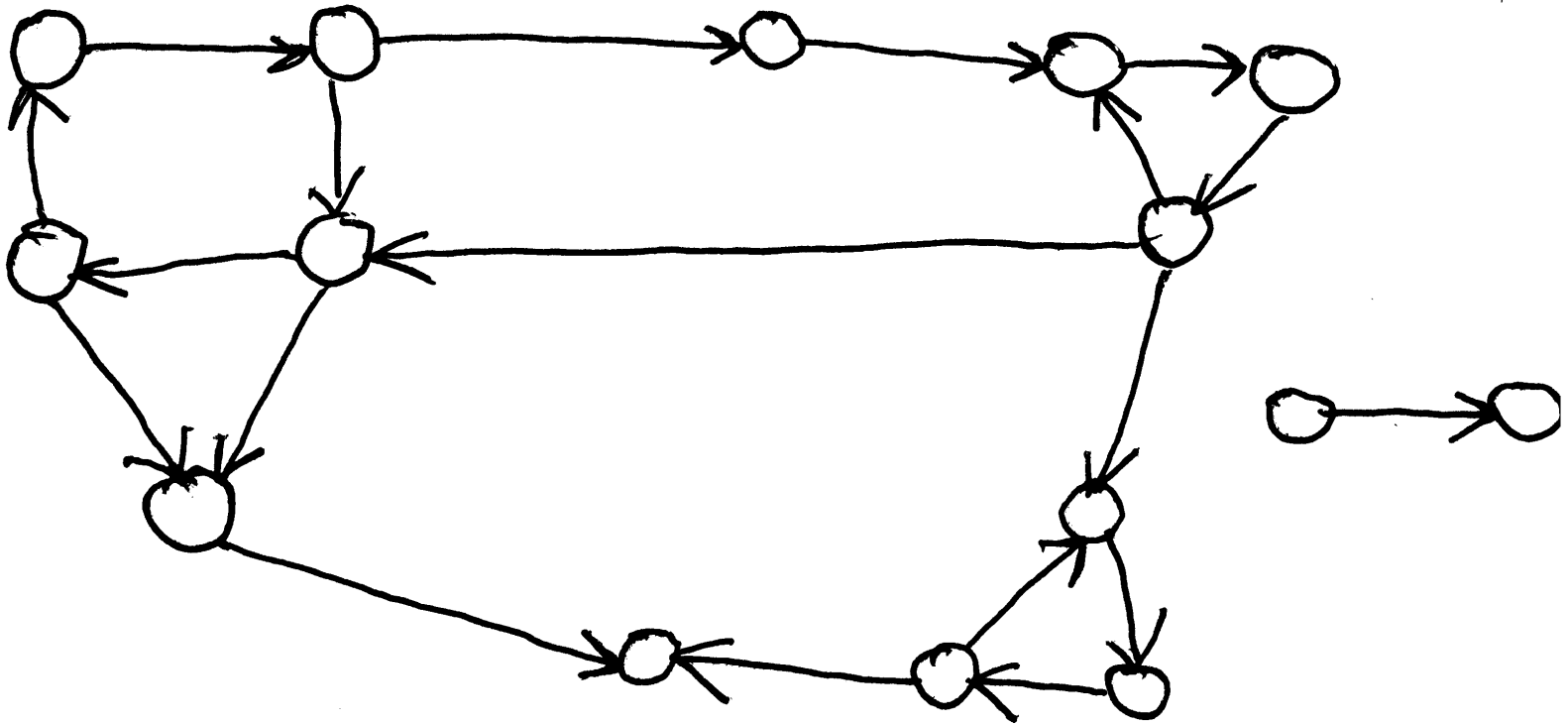
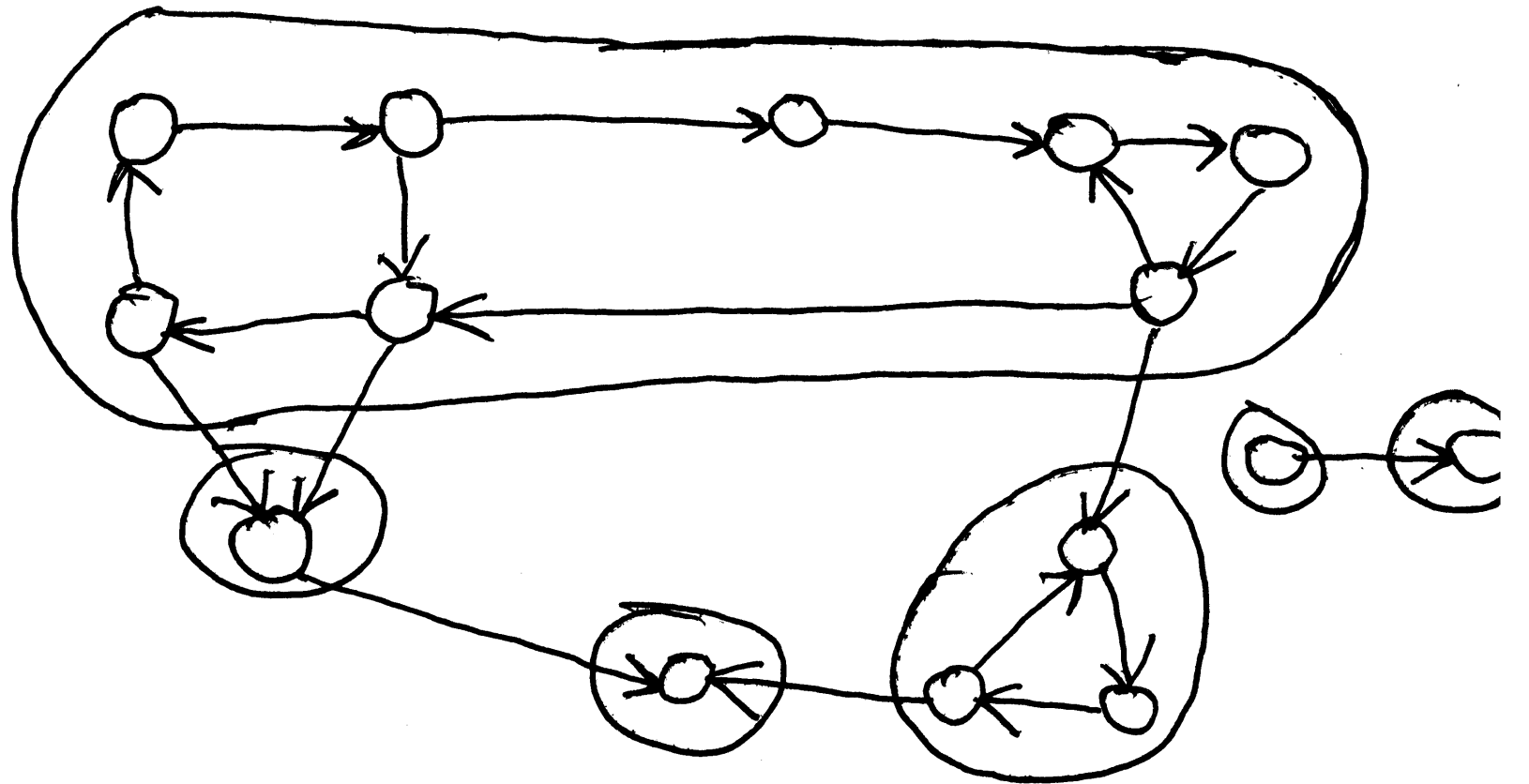

(More) Graphs

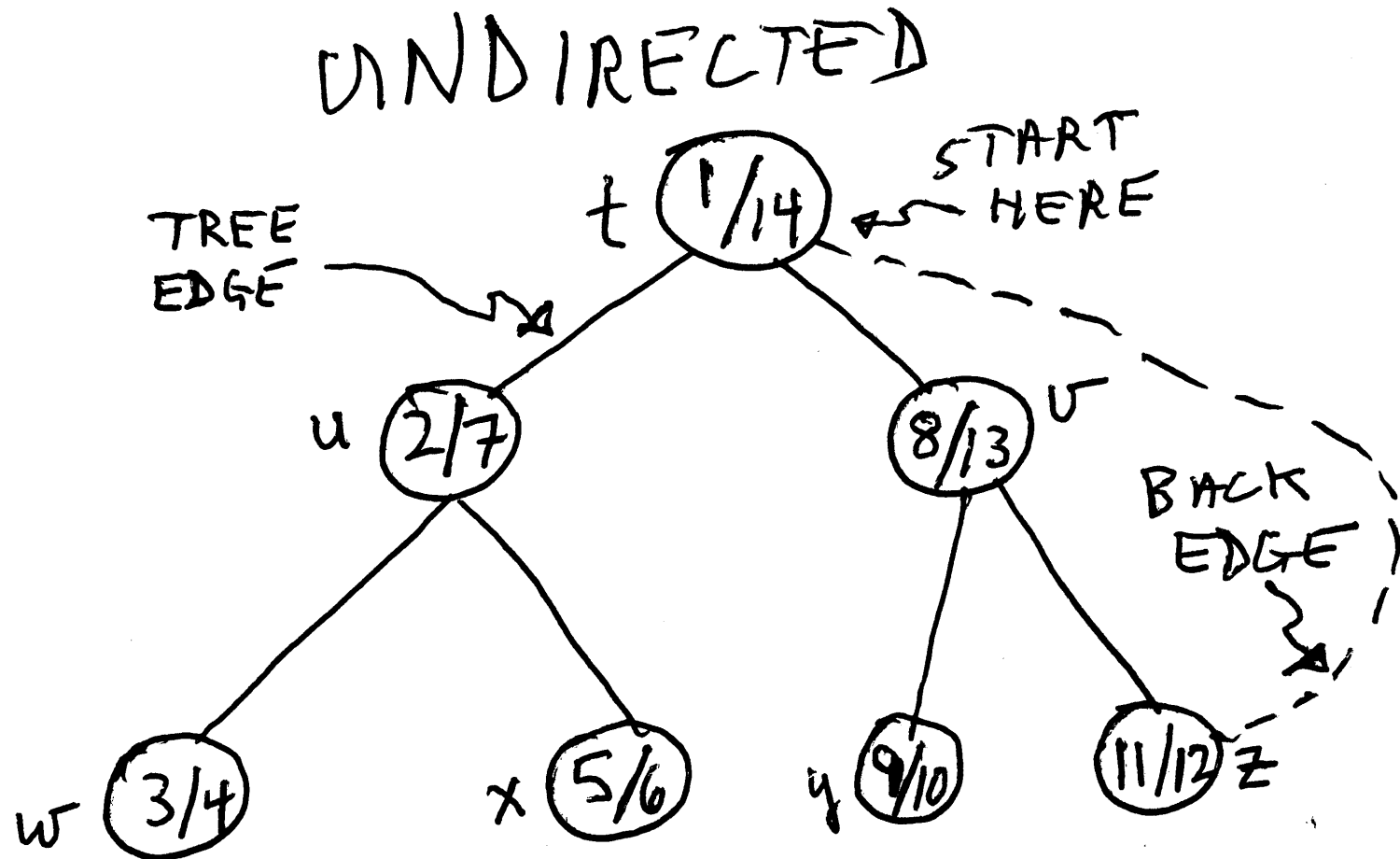
show strongly connected comps



see next slide



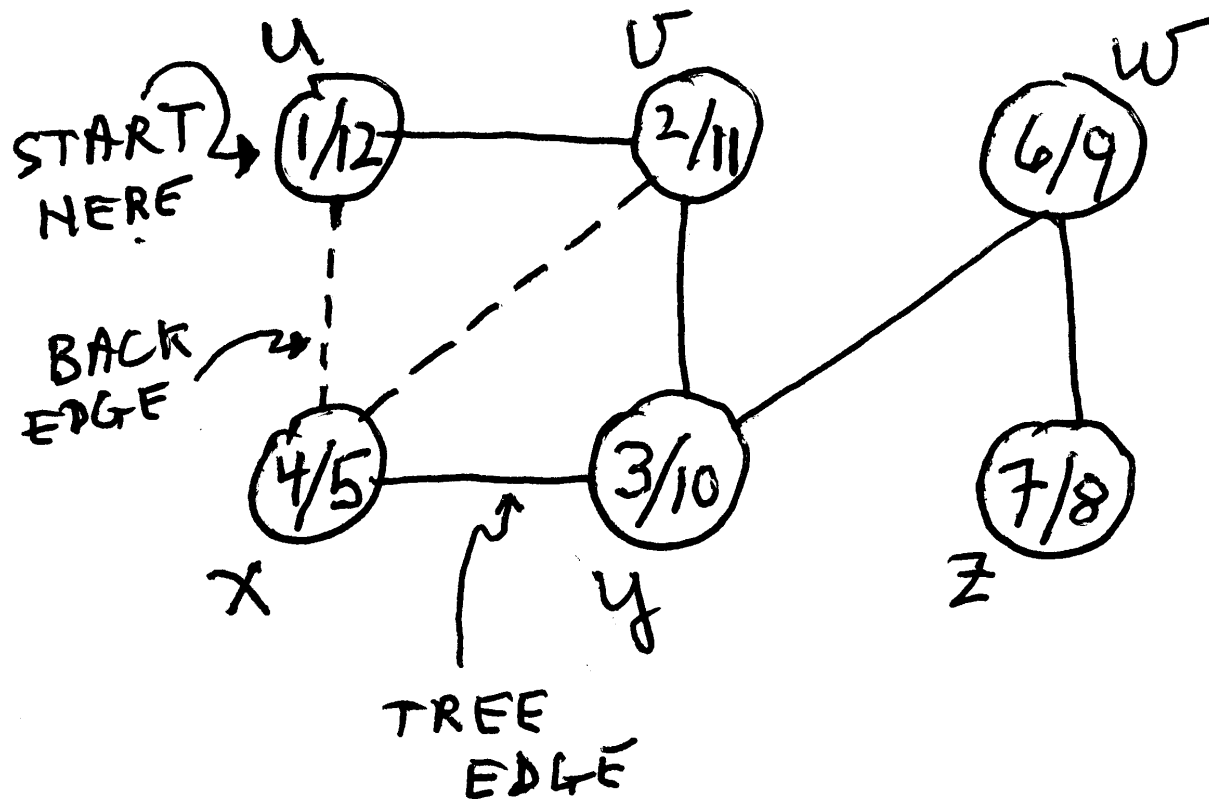
DFS examples



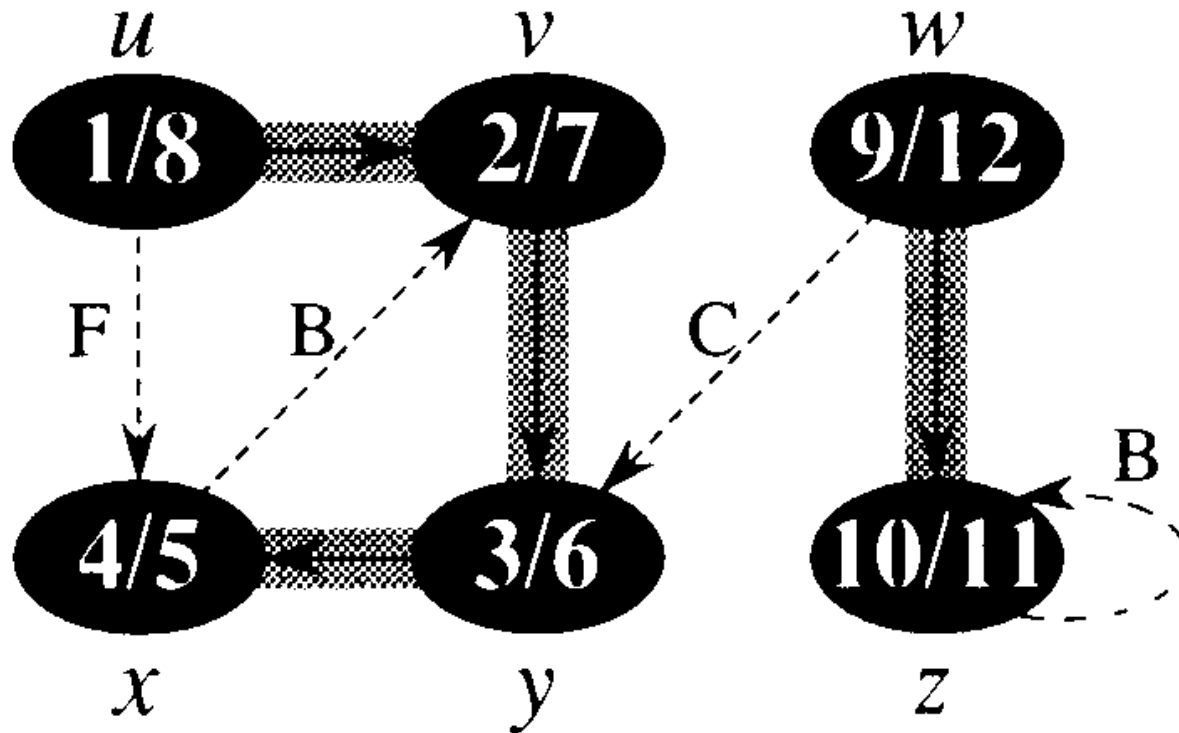
On an **undirected** graph, any edge that is not a “tree” edge is a “back” edge (from descendant to ancestor).

DFS Examples

UNDIRECTED



DFS Example: digraph



Here, we
get a
forest
(two
trees).

(p)

B = back edge (descendant to ancestor, or self-loop)

F = forward edge (ancestor to descendant)

C = cross edge (between branches of a tree, or between trees)

DFS running time is $\Theta(V+E)$

we visit each vertex once; we traverse each edge once

DFS(G)

```
1  for each vertex  $u \in V[G]$  }  $\Theta(V)$ 
2      do  $color[u] \leftarrow WHITE$ 
3       $\pi[u] \leftarrow NIL$ 
4   $time \leftarrow 0$ 
5  for each vertex  $u \in V[G]$ 
6      do if  $color[u] = WHITE$ 
7          then DFS-VISIT( $u$ )
```

CALLED ONLY ON WHITE VERTICES ($\Theta(V)$)

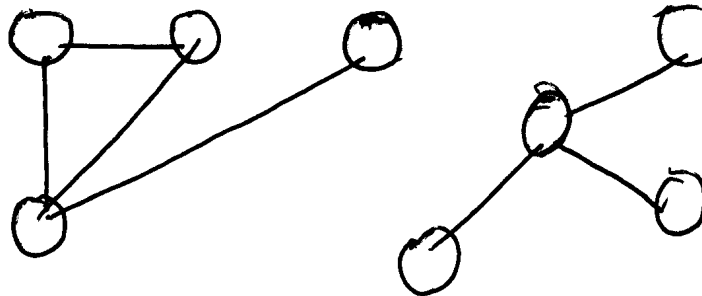
DFS-VISIT(u)

```
1   $color[u] \leftarrow GRAY$        $\triangleright$  White vertex  $u$  has just been discovered.
2   $time \leftarrow time + 1$ 
3   $d[u] \leftarrow time$ 
4  for each  $v \in Adj[u]$        $\triangleright$  Explore edge  $(u, v)$ .
5      do if  $color[v] = WHITE$ 
6          then  $\pi[v] \leftarrow u$ 
7          DFS-VISIT( $v$ )
8   $color[u] \leftarrow BLACK$      $\triangleright$  Blacken  $u$ ; it is finished.
9   $f[u] \leftarrow time \leftarrow time + 1$ 
```

$\sum_{u \in V} |Adj[u]| = \Theta(E)$

applications of DFS

Connected components of an **undirected** graph. Each call to DFS_VISIT (from DFS) explores an entire connected component (see ex. 22.3-11).



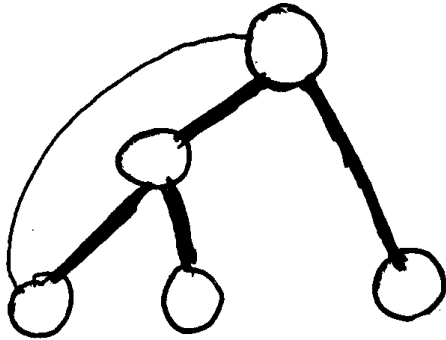
So modify DFS to count the number of times it calls DFS_VISIT:

```
5 for each vertex  $u \in V[G]$ 
6   do if color[u] = WHITE
6.5       then cc_counter  $\leftarrow$  cc_counter + 1
7           DFS_VISIT(u)
```

Note: it would be easy to label each vertex with its cc number, if we wanted to (i.e. add a field to each vertex that would tell us which conn comp it belongs to).

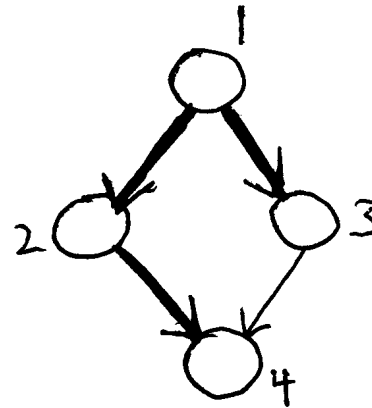
Applications of DFS

Cycle detection: Does a given graph G contain a cycle?



Idea #1: If DFS ever returns to a vertex it has visited, there is a cycle; otherwise, there isn't.

OK for **undirected** graphs, but what about:



No cycles, but a DFS from 1 will reach 4 twice.
Hint: what kind of edge is (3, 4)?

Cycle detection theorem

Theorem: A graph G (directed or not) contains a cycle if and only if a DFS of G yields a back edge.

→: Assume G contains a cycle. Let v be the first vertex reached on the cycle by a DFS of G . All the vertices reachable from v will be explored from v , including the vertex u that is just “before” v in the cycle. Since v is an ancestor of u , the edge (u, v) will be a **back edge**.

←: Say the DFS results in a back edge from u to v . Clearly, $u \rightarrow v$ (that should be a **wiggly arrow**, which means, “there is a path from u to v ”, or “ v is reachable from u ”). And since v is an ancestor of u (by def of back edge), $v \rightarrow u$ (again should be wiggly). So v and u must be part of a **cycle**. QED.

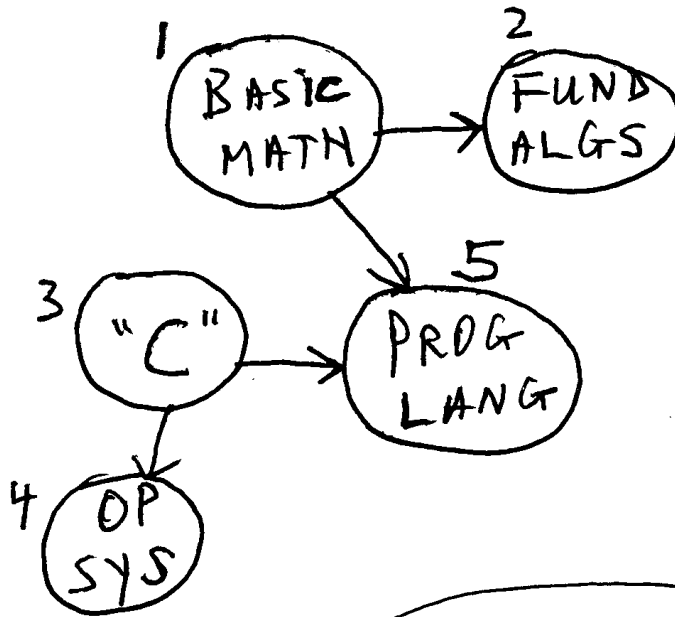
Back Edge Detection

How can we detect back edges with DFS? For **undirected** graphs, easy: see if we've visited the vertex before, i.e. *color* \neq WHITE.

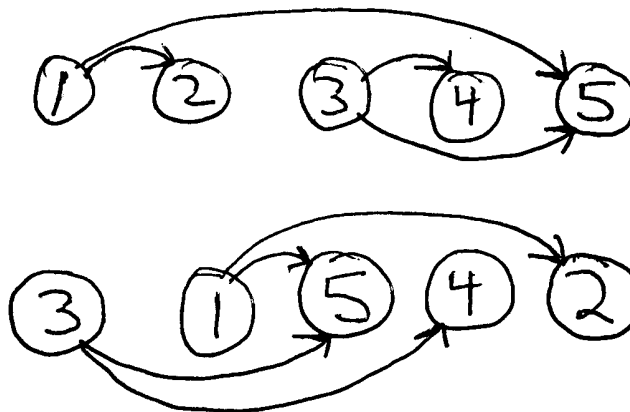
For **directed** graphs: Recall that we color a vertex GRAY while its adjacent vertices are being explored. If we re-visit the vertex while it is still GRAY, we have a back edge.

We blacken a vertex when its adjacency list has been examined completely. So any edges to a BLACK vertex cannot be back edges.

TOPOLOGICAL SORT



“Sort” the vertices so all edges go left to right.



TOPOLOGICAL SORT

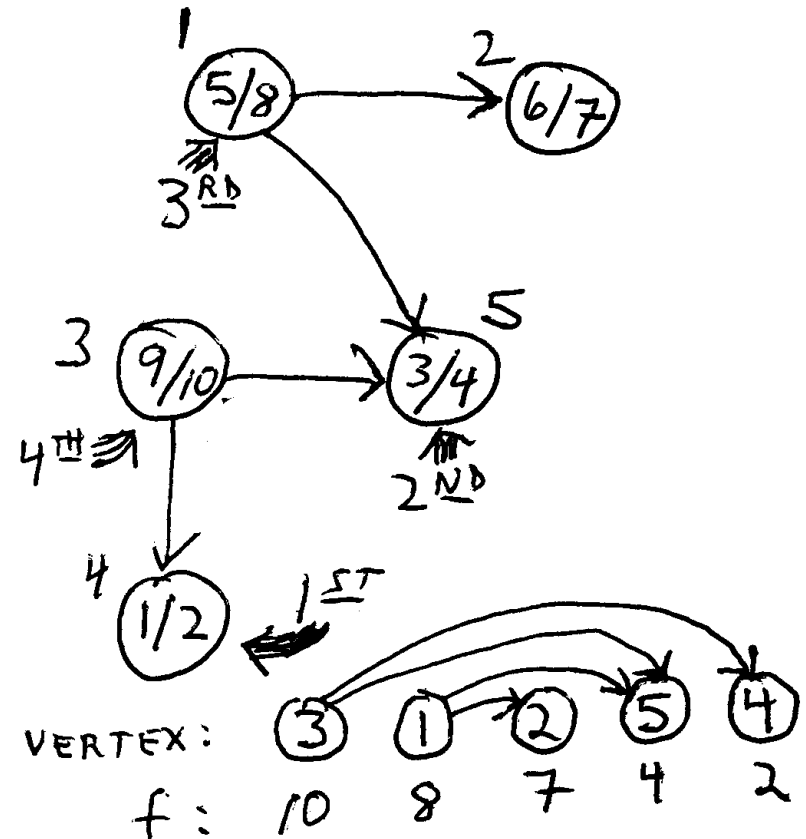
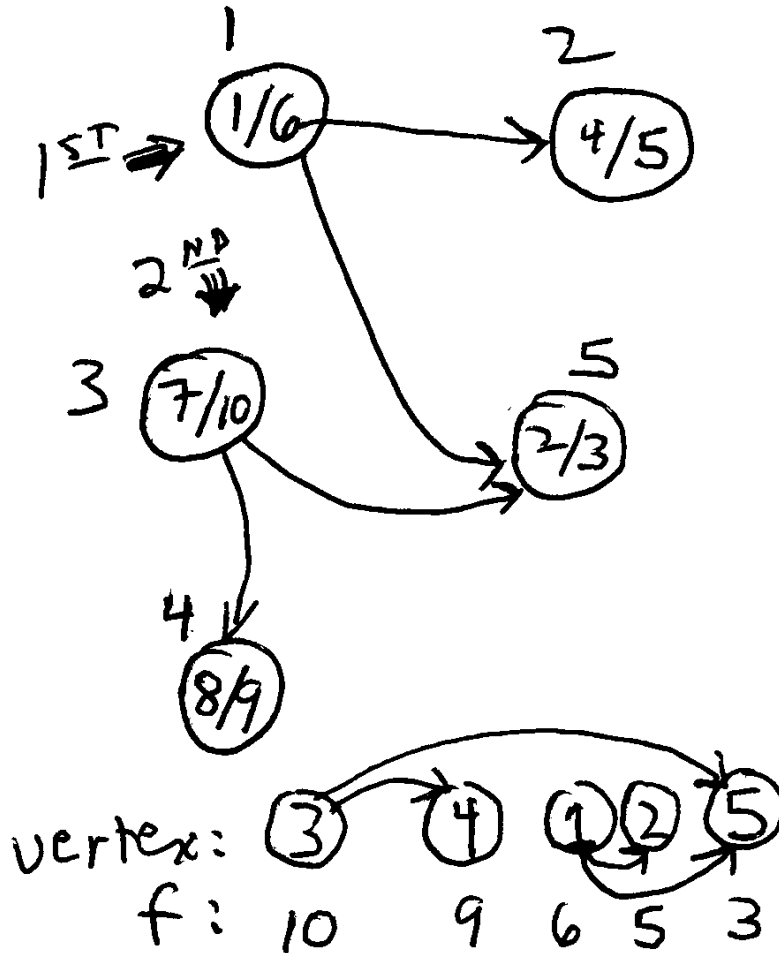
For topological sort to work, the graph **G** must be a **DAG** (directed acyclic graph). **G** 's undirected version (i.e. the version of **G** with the “directions” removed from the edges) need not be connected.

Theorem: Listing a dag's vertices in reverse order of finishing time (i.e. from highest to lowest) yields a topological sort.

Implementation: modify DFS to stick each vertex onto the front of a linked list as the vertex is finished.

see examples next
slide....

Topological Sort Examples



More on Topological Sort

Theorem (again): Listing a dag's vertices in order of highest to lowest finishing time results in a topological sort. Putting it another way: If there is an edge (u, v) , then $f[u] > f[v]$.

Proof : Assume there is an edge (u, v) .

Case 1: DFS visits u first. Then v will be visited and finished before u is finished, so $f[u] > f[v]$.

Case 2: DFS visits v first. There cannot be a path from v to u (why not?), so v will be finished before u is even discovered. So again, $f[u] > f[v]$.

QED.