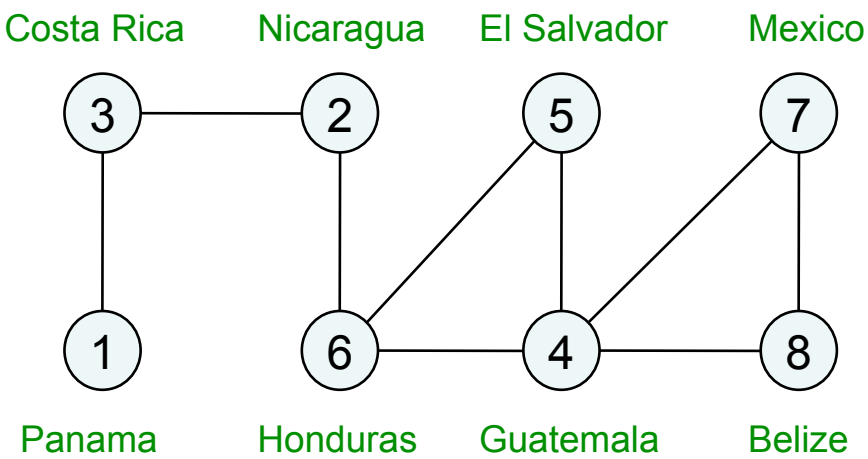
Graphs

A cartographer's problem



Graph specified by *nodes* and *edges*.

node	=	country
edge	=	neighbors

Graph coloring problem: color nodes of graph with as few colors as possible, so that there is no edge between nodes of the same color.

Exam scheduling

The registrar's problem

Pitchers (15)

#	Player	MWL Team	B	T	Ht.	Wt.	Born	Residence	'08 Stats (as of 6/30/08)
	Bryan Augenstein	South Bend	R	R	6-6	232	7/11/1986	Sebastian, FL	3-1, 2.09 ERA, 73.1 IP, 9BB
	Randy Boone	Lansing	R	R	6-3	215	8/6/1984	Yoakum, TX	6-2, 2.56 ERA, 59.2 IP, 13BB
	Mark Dapoules	Quad Cities	R	R	6-2	200	5/31/1988	Palm City, FL	4-0, 1.83 ERA, .226 BAA
	Edgar Estanga	Lansing	L	L	5-10	185	10/18/1985	Maturin Managas, VZ	5-1, 0.66 ERA, .182 BAA
	Alfredo Figaro	West Michigan	R	R	6-0	173	7/7/1984	Samanna, DR	7-2, 1.22 ERA, .176 BAA
	Jeff Jeffords	Dayton	R	R	6-1	200	11/4/1984	Lamar, SC	2-1, 2.83 ERA, 28.2 IP, 37K
	Jon Kibler	West Michigan	L	L	6-4	215	8/10/1986	Freeland, MD	5-2, 2.20 ERA, .159 BAA
	Steven Johnson	Great Lakes	R	R	6-1	212	8/31/1987	Kingsville, MD	6-2, 2.60 ERA, .214 BAA
	Joseph Krebs	Dayton	L	L	6-0	200	9/14/1984	Bridgeport, TX	5-2, 2.43 ERA, 4 saves
	Derek McDaid	Fort Wayne	R	R	6-1	200	9/27/1983	Barrie, ON, Canada	3-0, 2.52 ERA, .225 BAA
	Brad Mills	Lansing	L	L	6-0	185	3/5/1985	Mesa, AZ	4-2, 2.62 ERA, 58.1IP, 68K
	Luis Montano	Dayton	R	R	6-0	180	3/20/1985	Santo Domingo, DR	6-3, 4.45 ERA, 56.2 IP, 13BB
	Jarrod Parker	South Bend	R	R	6-1	190	11/24/1988	Ossian, IN	4-2, 2.45 ERA, .236 BAA
	Miguel Ramirez	Great Lakes	R	R	5-11	165	7/15/1983	Fondo Negro, DR	1-3, 0.37 ERA, 12 saves
	Evan Scribner	South Bend	R	R	6-3	190	7/19/1985	New Britain, CT	2-3, 2.05 ERA, 26.1IP, 40K

Catchers (2)

#	Player	MWL Team	B	T	Ht.	Wt.	Born	Residence	'08 Stats (as of 6/30/08)
	Sean Coughlin^	South Bend	L	R	6-1	206	5/14/1985	Morrison, CO	.265, 6 HR, 24 RBI, .541SLG
	Kenley Jansen	Great Lakes	B	R	6-4	225	9/30/1987	Curacao, Netherlands Antilles	.204, 5 HR, 11 RBI

Infielders (11)

#	Player	MWL Team	B	T	Ht.	Wt.	Born	Residence	'08 Stats (as of 6/30/08)
	Kevin Ahrens	Lansing	B	R	6-1	190	4/26/1989	Houston, TX	.260, HR, 24 RBI, 14 doubles
	Chris Carlson^	West Michigan	R	R	6-4	225	1/7/1984	Topeka, KS	.291, 7 HR, 32 RBI
	Felix Carrasco	Fort Wayne	B	R	6-1	244	2/14/1987	Bani, DR	.255, 9 HR, 37 RBI
	Justin Jackson	Lansing	R	R	6-2	175	12/11/1988	Asheville, NC	.250, 3 HR, 23 RBI, 40 R
	Pete Kozma*	Quad Cities	R	R	6-0	170	4/11/1988	Owasso, OK	.274, 3 HR, 19 RBI
	Mike Mee	South Bend	L	R	6-0	188	10/14/1983	Richfield, MN	.260, 2 HR, 21 RBI, .379 OBP
	Andy Parrino*	Fort Wayne	B	R	6-0	177	10/31/1985	Brockport, NY	.276, 3 HR, 15 RBI, .387 OBP
	Manny Rodriguez*	Lansing	L	L	6-3	190	1/6/1985	Chitre, Panama	.310, 4 HR, 37 RBI,19 doubles
	John Tolisano	Lansing	B	R	5-11	180	10/7/1988	Estero, FL	.274, HR, 23 RBI, 6 triples
	Joe Tucker	West Michigan	R	R	5-11	170	1/25/1984	Canton, OH	.273, 15 RBI, 13 K in 132 AB
	Brandon Waring*	Dayton	R	R	6-4	195	1/2/1986	West Columbia, SC	.267, 11 HR, 32 RBI, .497 SLG

Outfielders (5)

#	Player	MWL Team	B	T	Ht.	Wt.	Born	Residence	'08 Stats (as of 6/30/08)
	Evan Frey*	South Bend	L	L	5-11	171	6/7/1986	Edwardsville, IL	.333, 22 RBI, 13 SB, .384 OBP
	Charlie Kingrey*	Quad Cities	L	L	6-2	210	1/19/1985	Kinder, LA	.308, 5 HR, 32 RBI,15 doubles
	Andrew Lambo*	Great Lakes	L	L	6-3	200	8/11/1988	Newbury Park, CA	.267, 7 HR, 41 RBI,14 doubles
	Denis Phipps	Dayton	R	R	6-2	176	7/22/1985	San Pedro de Macoris, DR	.261, 4 HR, 29 RBI,12 doubles
	Casper Wells	West Michigan	R	R	6-2	210	11/23/1984	Schenectady, NY	.237, 10 HR, 26 RBI, 15 SB

Schedule final exams:

- use as few time slots as possible

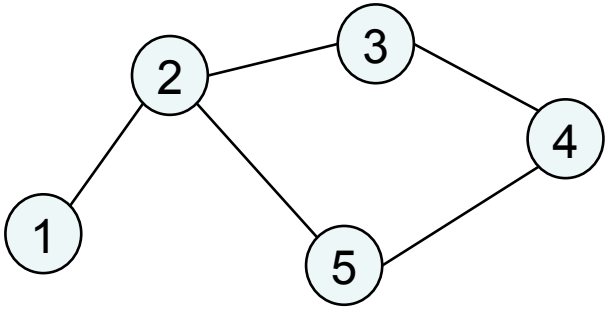
- can't schedule two exams in the same slot if there's a student taking both classes.

This is also graph coloring!

Node = exam

Edge = some student is taking both endpoint-exams

Color = time slot

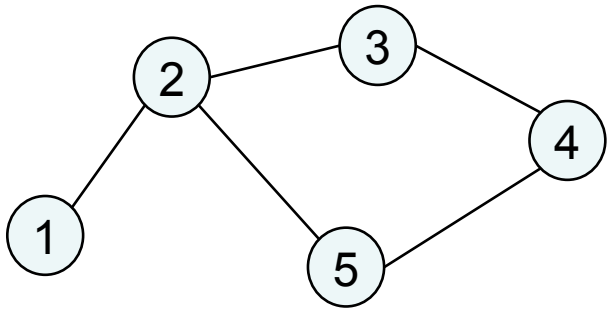


Graphs, formally

G = (V,E) where

V: vertices/nodes

E: edges



V = {1,2,3,4,5}

E = {{1,2}, {2,3}, {3,4}, {2,5}, {4,5}}

Undirected edges: symmetric relationship

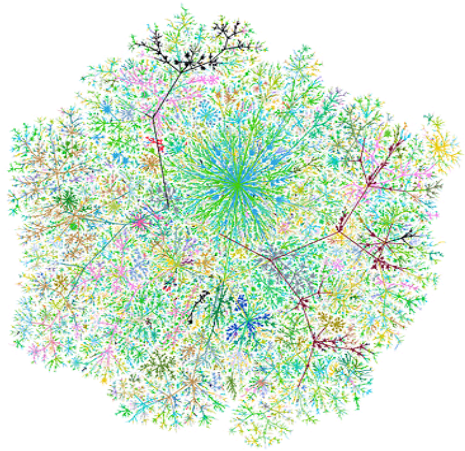
Directed graphs

(x,y): edge *from* x *to* y

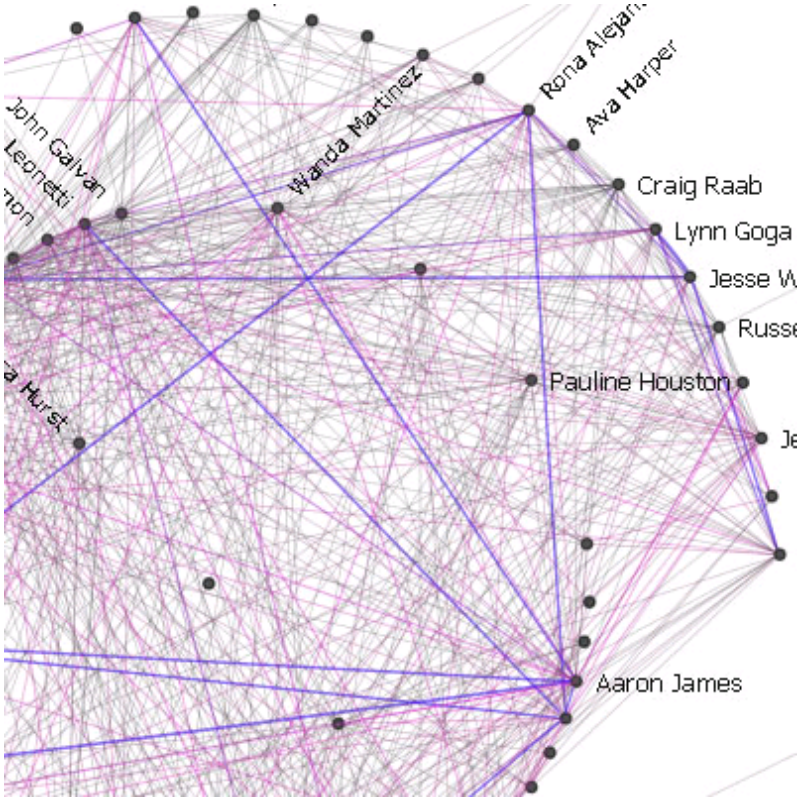
e.g.World wide web

node	URL
edge (u,v)	u points to v

Billions of nodes and edges!



Social networks



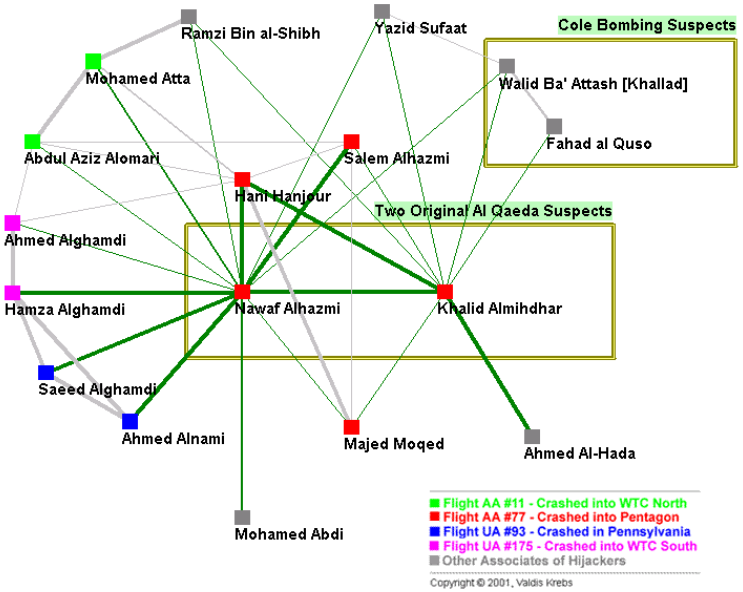
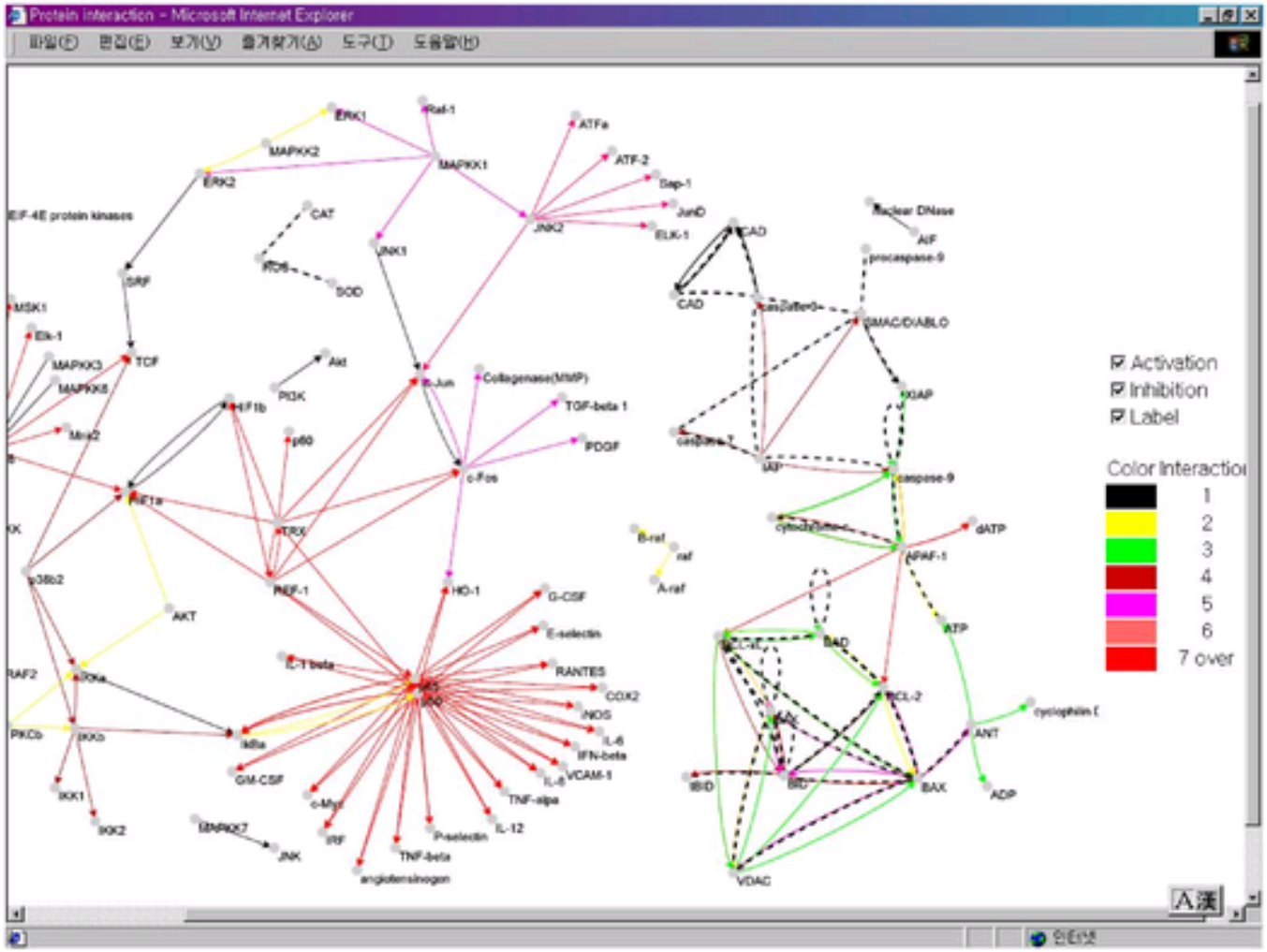


Figure 2 - All nodes within 1 step [direct link] of original suspects

Biological networks



How are graphs stored on a computer?

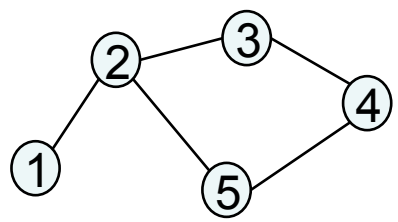
Adjacency matrix

V x V matrix A

A(i,j) = 1 if (i,j) is in E
0 otherwise

Symmetric if G undirected

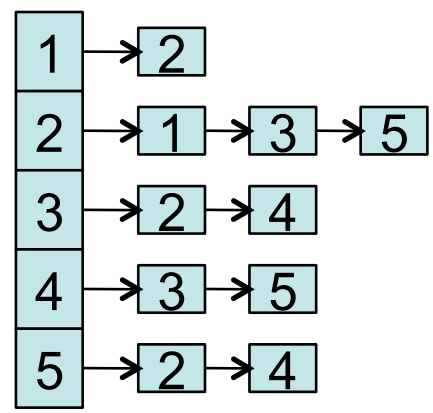
$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \end{pmatrix}$$



PRO check for an edge in O(1) time
CON uses up $O(V^2)$ space

Adjacency list

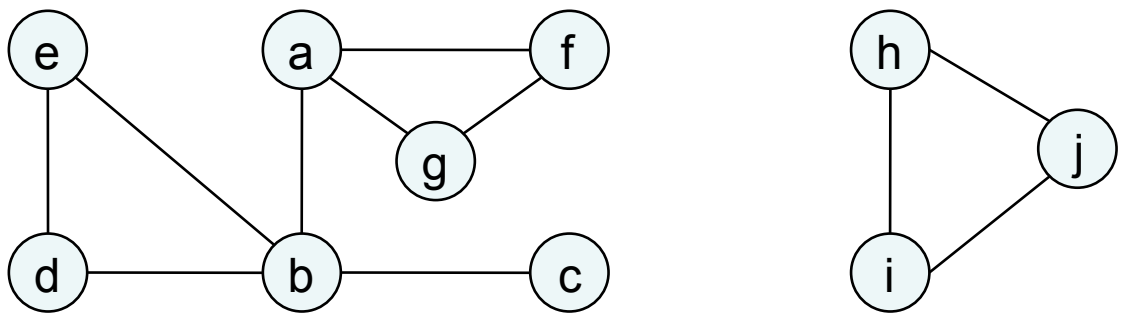
For each node, list of outgoing edges



PRO just O(E) space
CON check for an edge in O(V) time
PRO easily iterate through node's neighbors

Depth-first search in undirected graphs

What parts of a graph are reachable from a given vertex?



With an adjacency list representation, this is like navigating a maze...

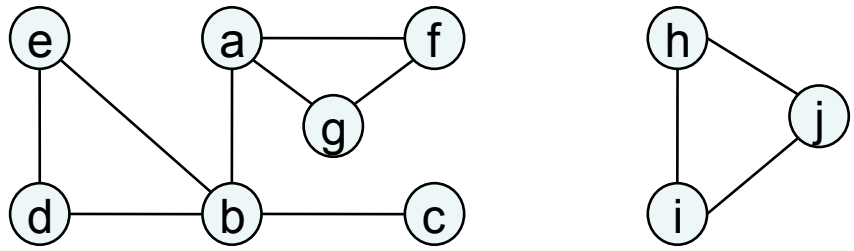
Potential difficulty	**Don't go round in circles**	**Don't miss anything**
Classical solution	Piece of chalk to mark visited junctions	Ball of string – leads back to starting point
Cyber-analog	Boolean variable for each vertex: visited or not	STACK

An exploration procedure

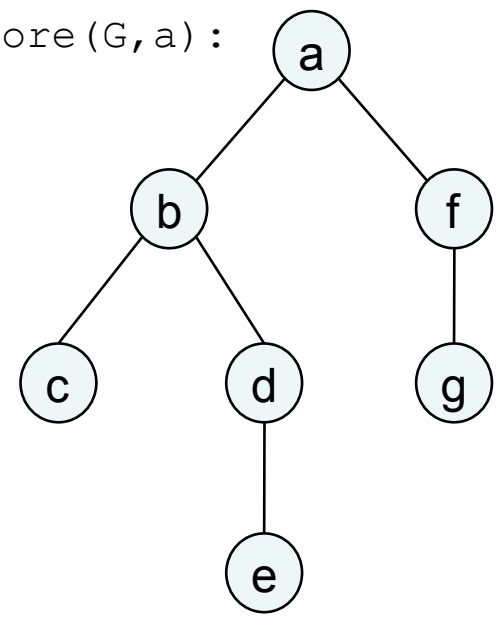
```
procedure explore(G,v)

input: graph G = (V,E); node v in V
output: visited[u] is set to true
   for all u reachable from v

visited[v] = true
previsit(v)
for each edge (v,u) in E:
   if not visited[u]:
        explore(G,u)
postvisit(v)
```



explore(G,a):



Does “explore” work?

```
procedure explore(G,v)
visited[v] = true
for each edge (v,u) in E:
   if not visited[u]:
        explore(G,u)
```

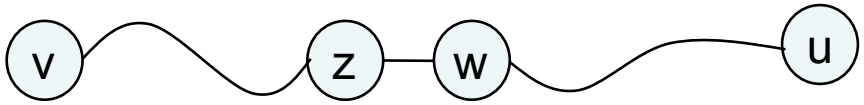
Does it actually halt?

For any node u, explore(G,u) is called at most once;
thereafter visited[u] is set.

Does it visit everything reachable from v?

Suppose it misses node u reachable from v; we’ll derive a contradiction.

Pick any path from v to u, and let z be the last node on the path that was visited.



But w would not have been overlooked during explore(G,z); this is a contradiction.

Alternative proof

```
procedure explore(G,v)
visited[v] = true
for each edge (v,u) in E:
   if not visited[u]:
        explore(G,u)
```

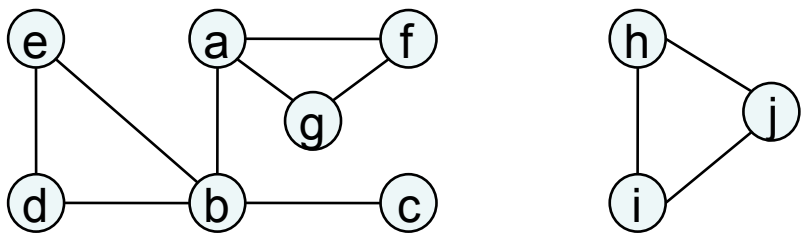
Does explore(G,v) visit everything reachable from v?

Do a proof by induction.

Undirected connectivity

An undirected graph is *connected* if there is a path between any pair of nodes.

```
procedure dfs(G)
for all v in V:
   visited[v] = false
for all v in V:
   if not visited[v]:
        explore(G,v)
```

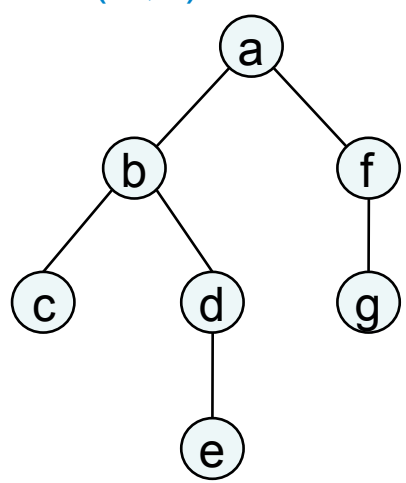


This graph has 2 *connected components*.

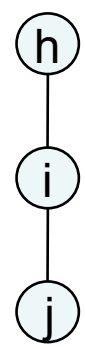
explore(G,v) returns the connected component containing v.
To explore the rest of the graph, restart explore() elsewhere.

DFS decomposes an undirected graph into its connected components!

explore(G,a)



explore(G,h)



Running time analysis

```
procedure explore(G,v)
visited[v] = true
for each edge (v,u) in E:
   if not visited[u]:
        explore(G,u)
```

```
procedure dfs(G)
for all v in V:
   visited[v] = false
for all v in V:
   if not visited[v]:
        explore(G,v)
```

How long does dfs(G) take?

explore(G,v) is called exactly once for each node v.

During this call,
time = O(1) + time for inner loop

Therefore total time =
O(V) + time for inner loops

During inner loops: each edge is examined twice, once from each endpoint. Therefore O(E).

Total: O(V+E), linear in the size of the graph.

Alternative running time analysis

```
procedure explore(G,v)
visited[v] = true
for each edge (v,u) in E:
   if not visited[u]:
        explore(G,u)
```

```
procedure dfs(G)
for all v in V:
   visited[v] = false
for all v in V:
   if not visited[v]:
        explore(G,v)
```

How long does dfs(G) take?

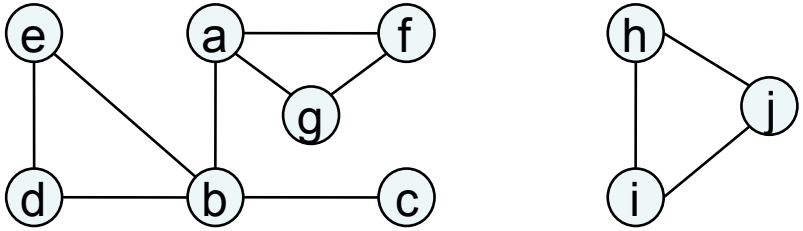
explore(G,v) is called exactly once for each node v.

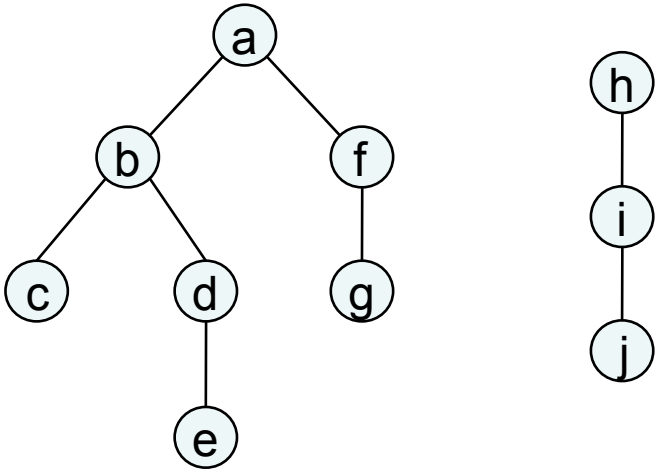
Pre- and post-visit numbers

```
procedure explore(G,v)
visited[v] = true
previsit(v)
for each edge (v,u) in E:
   if not visited[u]:
        explore(G,u)
postvisit(v)
```

```
procedure dfs(G)
for all v in V:
   visited[v] = false
for all v in V:
   if not visited[v]:
        explore(G,v)
```

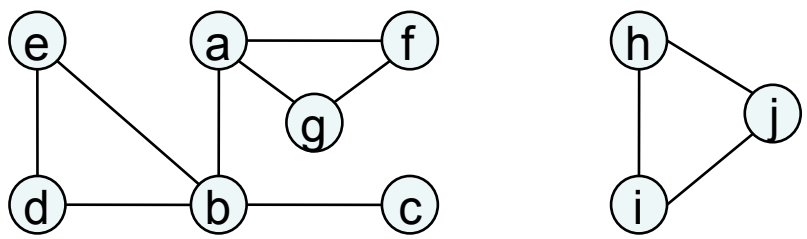
```
procedure previsit(v)
pre[v] = clock++
procedure postvisit(v)
post[v] = clock++
```





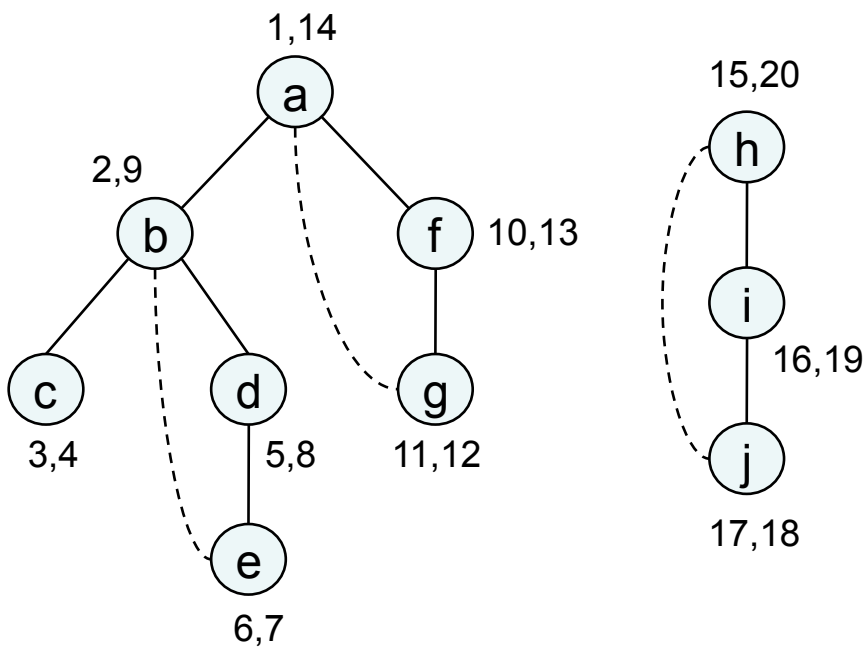
Extra information to record:
pre[u] = time of initial discovery
post[u] = time of final departure

Undirected DFS: wrap-up



The intervals [pre[u], post[u]] are either nested or disjoint. Why?

[pre[u],post[u]] is the time when node u is on the stack.



Terminology:
DFS search forest consisting of two *DFS search trees*

——— *tree edge*: traversed by DFS

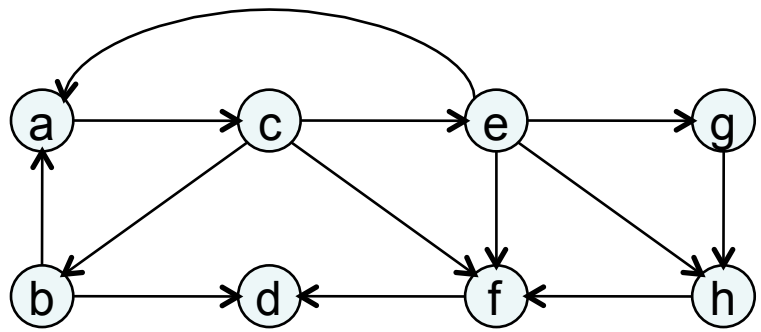
- - - - - - - *back edge*: not traversed (led to a node already visited)

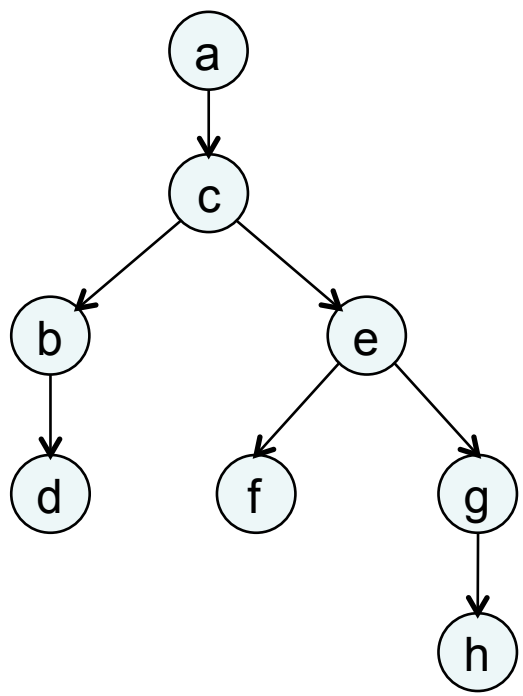
Directed DFS: example

```
procedure explore(G,v)
visited[v] = true
previsit(v)
for each edge (v,u) in E:
   if not visited[u]:
        explore(G,u)
postvisit(v)
```

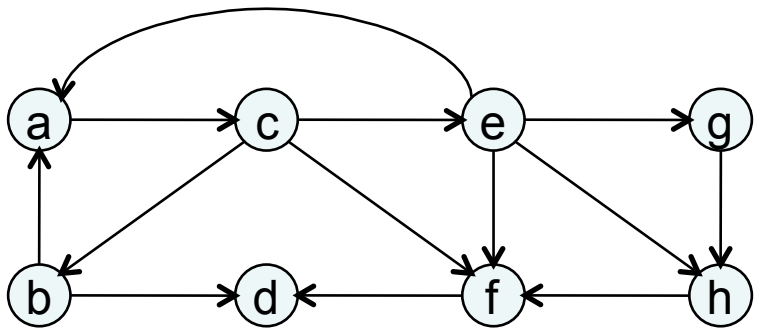
```
procedure dfs(G)
for all v in V:
   visited[v] = false
for all v in V:
   if not visited[v]:
        explore(G,v)
```

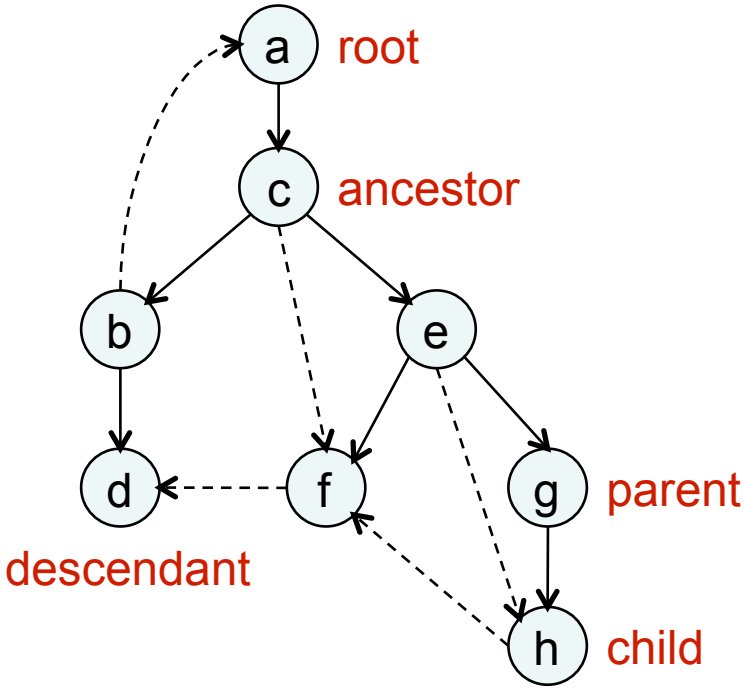
```
procedure previsit(v)
pre[v] = clock++
procedure postvisit(v)
post[v] = clock++
```





Directed DFS: terminology

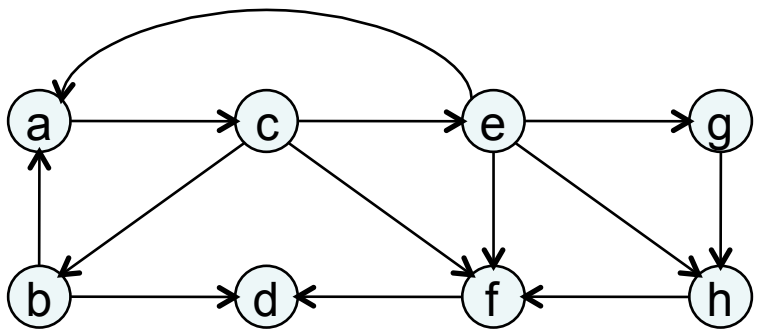




Four types of edges

tree edge	part of DFS forest
back edge	leads to an ancestor
forward edge	leads to non-child descendant
cross edge	leads to neither descendant nor ancestor

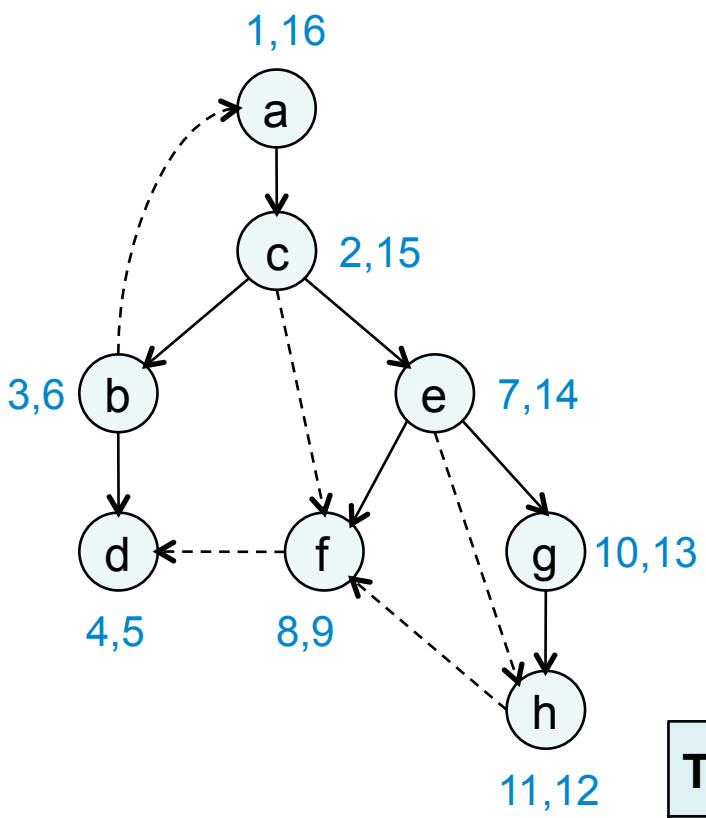
Directed DFS: example



Four types of edges

tree edge	part of DFS forest
back edge	leads to an ancestor
forward edge	leads to non-child descendant
cross edge	leads to neither descendant nor ancestor

The pre/post signature of ancestors



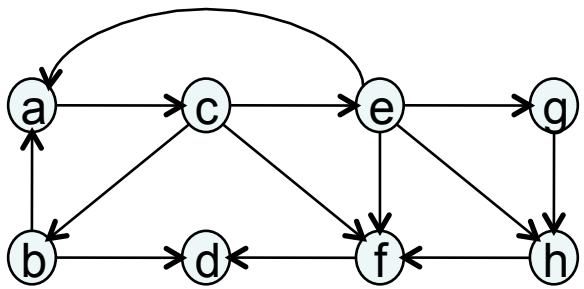
Node u is an ancestor of node v
if and only if
pre[u] < pre[v] < post[u]

Why? Because:
u is an ancestor of v
if and only if
u is discovered first AND
v is discovered during the
exploration of u

Type of edge	pre/post criterion for edge (u,v)
Tree	pre[u] < pre[v] < post[v] < post[u]
Forward	pre[u] < pre[v] < post[v] < post[u]
Back	pre[v] < pre[u] < post[u] < post[v]
Cross	pre[v] < post[v] < pre[u] < post[u]

Cycles

A *cycle* in a directed graph is a circular path $v_0 \to v_1 \to \cdots \to v_k \to v_0$



Graph without cycles: *acyclic*.

How to tell if a directed graph is acyclic?

<u>Claim</u> A directed graph G has a cycle if and only if DFS encounters a back edge.

(() Suppose DFS encounters a back edge from node v to node u.
Then G has a cycle consisting of the path from u to v in the search tree, plus edge (v,u).

()) Suppose G has a cycle

$$v_0 \to v_1 \to \cdots \to v_k \to v_0$$

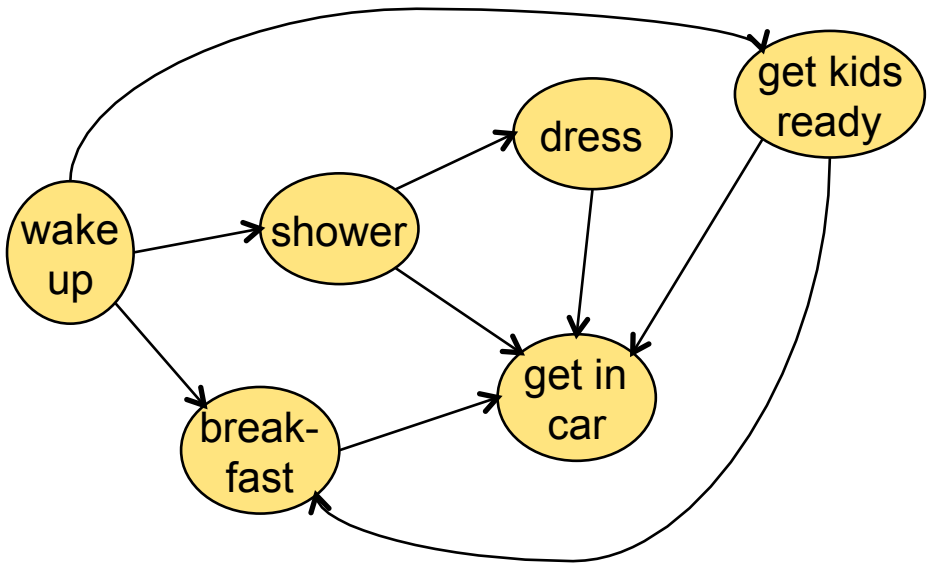
Let v_i be the first of these nodes to be explored; then the rest of them lie in the DFS subtree below v_i; and (v_{i-1}, v_i) (or (v_k, v_0) if i=0) is a back edge.

Linear time algorithm to check acyclicity!

Directed acyclic graphs (dags)

For modeling hierarchy, causality, temporal dependency,...

Scheduling problem:



In what order should tasks be performed?
If there is a cycle: no hope!

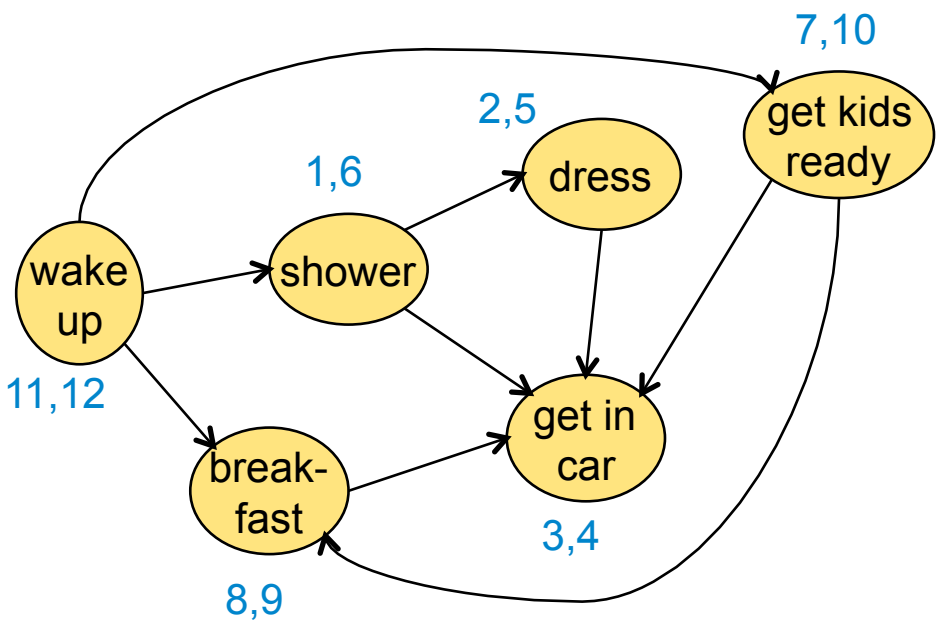
Topological ordering

Input: a dag
Goal: give each node a number so that every edge leads from a lower number to a higher number (i.e. precedence constraints satisfied).

Solution:
Run DFS and perform tasks in order of decreasing POST numbers.

Claim In a dag, every edge leads to a lower post number.
Proof: The only edges (u,v) for which post[v] > post[u] are back edges.
And a dag has no back edges!

DAGs, cont'd



A source is a node with no in-edges.
A sink is a node with no out-edges.

<u>Claim</u> In a dag, the node with highest post number is a source and lowest post number is a sink.

Another algorithm for topological ordering:
- Find a source, output it
- Delete it from the graph
- Repeat until graph is empty

Topological sorting, method 2

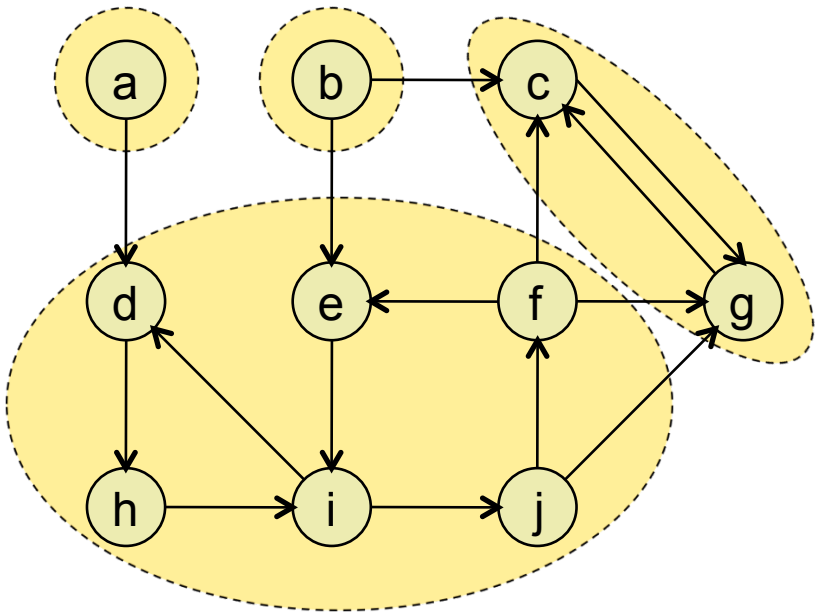
- Find a source, output it
- Delete it from the graph
- Repeat until graph is empty

Topological sorting, method 2

Justification of correctness

Running time analysis

Connectivity in directed graphs



In directed graphs, we say

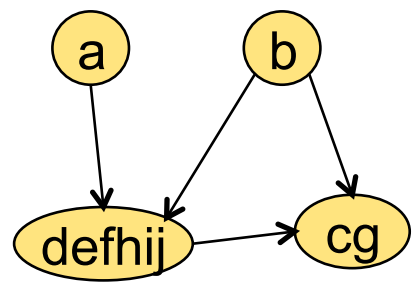
u is connected to v

if there is a path from u to v AND from v to u.

Partition V into strongly connected components.

The metagraph

Shrink each SCC to a *meta-node*.
Put an edge from one meta-node to another if there is an edge (in the same direction) between their respective vertices.



2 source SCCs
1 sink SCC

Every directed graph is the DAG of its strongly connected components.

Two-tiered structure of directed graph:
Top level: DAG, very simple structure
Finer detail: peek inside one of the meta-nodes

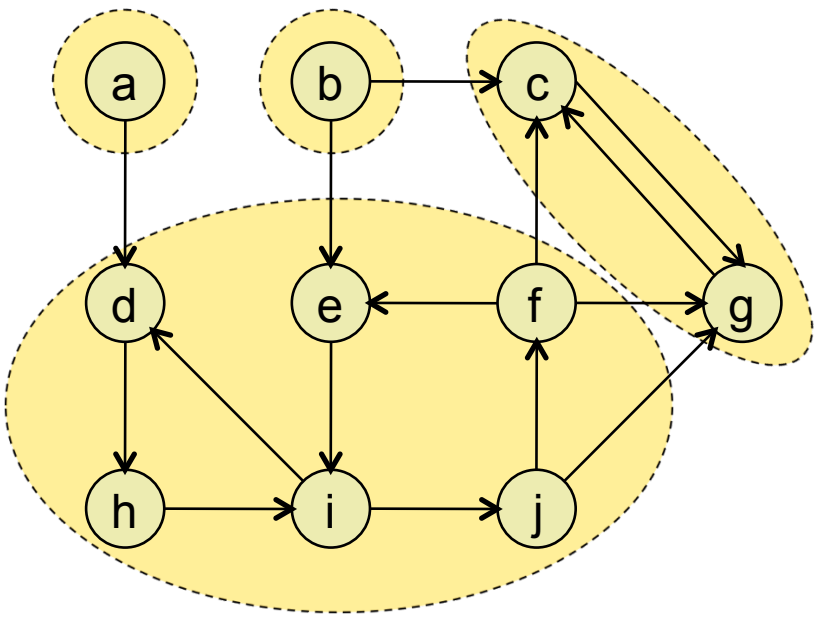
Decomposing a graph into its SCCs

<u>Property 1</u>: If the *explore* subroutine is started at node u, it will terminate when all nodes reachable from u have been visited.

So: if we start in a sink SCC, we will precisely identify that SCC!

Two problems:

A. How to find a node that is guaranteed to be in a sink SCC?
B. Once we've identified a sink SCC, how do we continue?

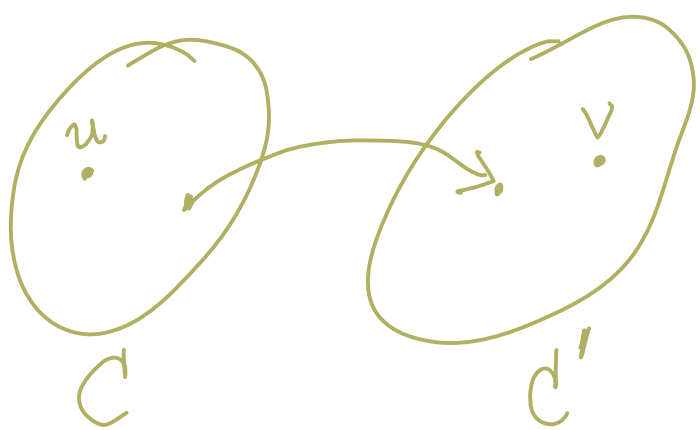


Problem (A): we can always find a node that is guaranteed to be in a source SCC!

Finding a node in a source SCC

Property 2: Run DFS on G. The node with the highest post number lies in a source SCC.

Follows from:
Property 3: If C, C' are SCCs and there is an edge from C to C' then the highest post number in C is bigger than the highest post number in C'.



Case 1: DFS sees C first

Suppose DFS first sees node u in C.
Then it sees all of C' while exploring u.
Therefore post[u] is bigger than every post number in C'.

Case 2: DFS sees C' first

Suppose DFS first sees node v in C'.
Then it sees all of C' while exploring v, but none of C.
Therefore every post number in C' is less than any post number in C.

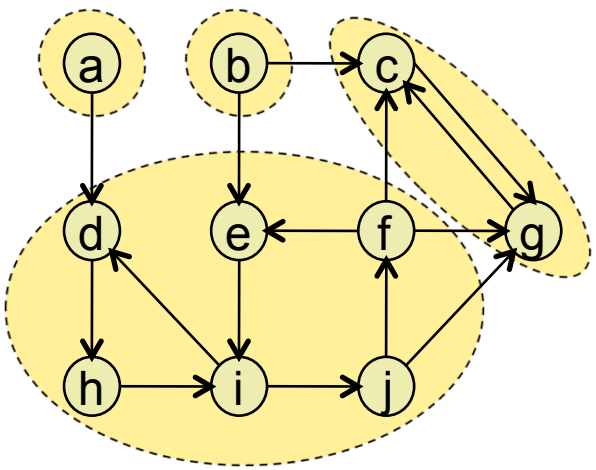
The SCCs can be topologically sorted by arranging them in decreasing order of their highest post numbers.

Decomposing a graph into its SCCs

Property 1: *explore(G,u)* terminates when all nodes reachable from u have been visited.

So: if we start in a sink SCC, we will precisely identify that SCC!

A. How to find a node that is guaranteed to be in a sink SCC?
B. Once we've identified a sink SCC, how do we continue?



Problem (A)

We can always find a node that is guaranteed to be in a source SCC.

Reverse graph G^R = same as G, with edges reversed

G^R has the same SCCs as G

Source SCC in G^R = sink SCC in G

Therefore: run DFS on G^R and pick node with highest post number; this lies in a sink SCC of G.

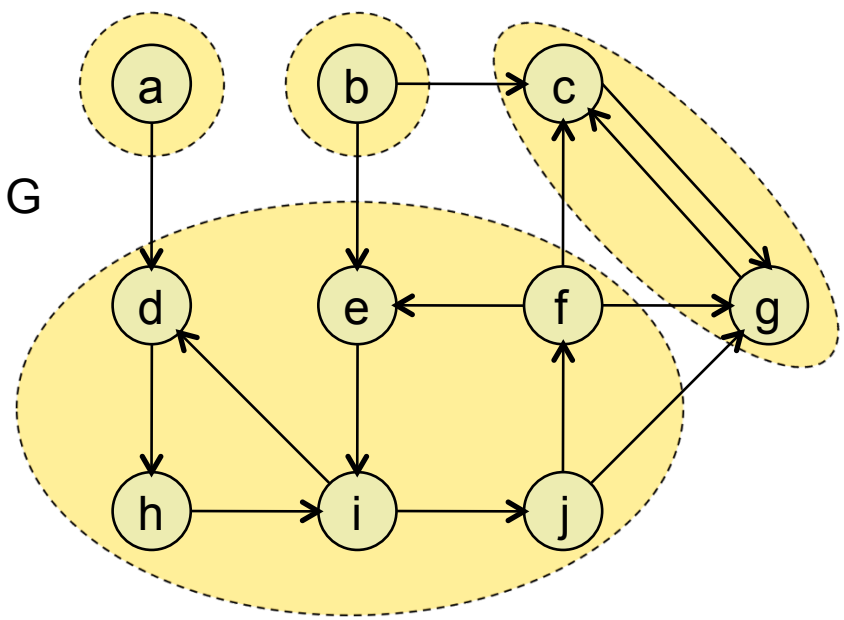
Problem (B)

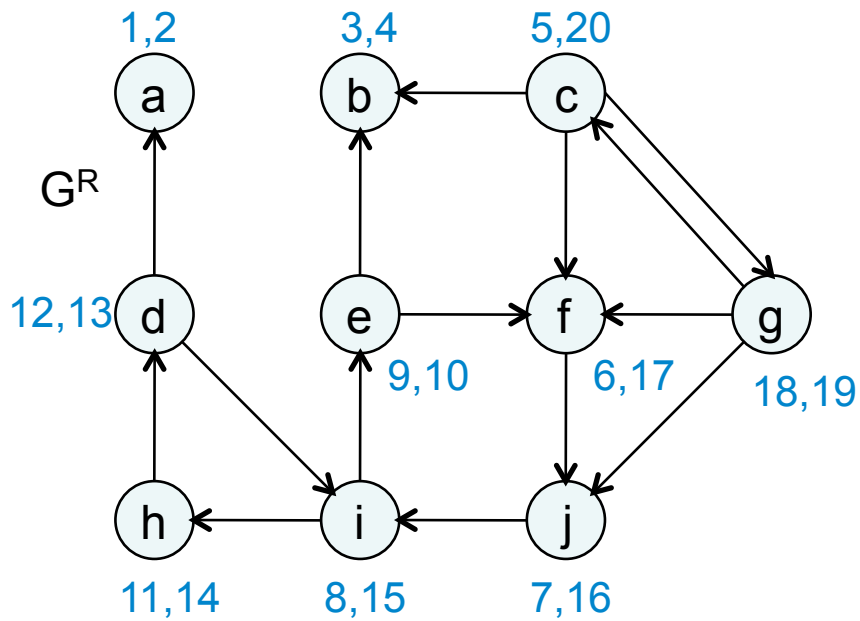
Identify sink SCC, delete from graph.

Of the remaining nodes, the one with highest post number (in G^R) will be in a sink SCC of whatever is left of G.

SCC algorithm

```
run DFS on G^R
for v in V, in decreasing order of G^R-post numbers:
    if not visited[v]:
        explore(G,v)
        output nodes seen as a SCC
```





Ordering from G^R: c,g,f,j,i,h,d,e,b,a