Lecture #5

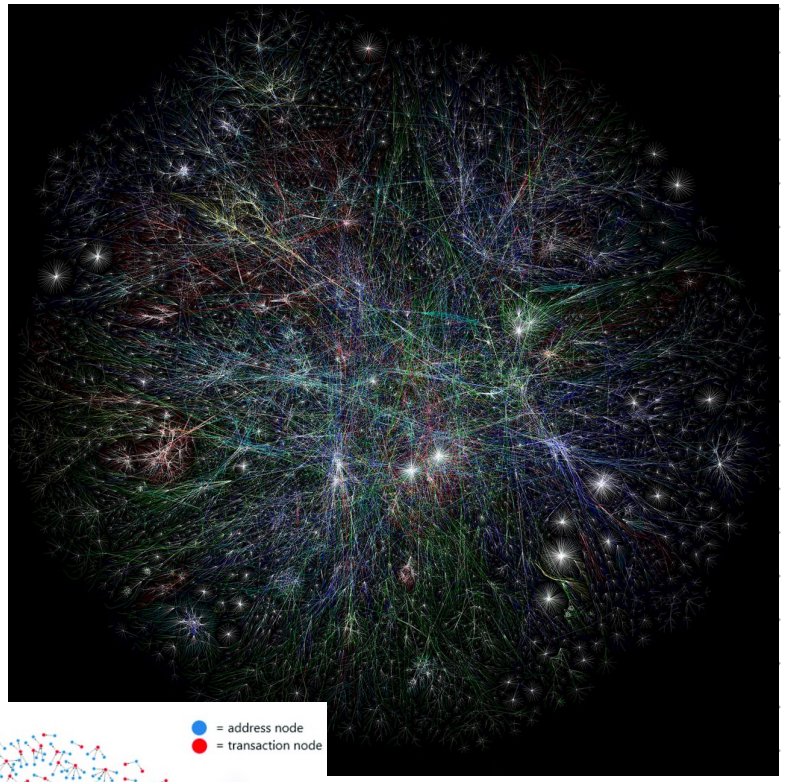
CS 170

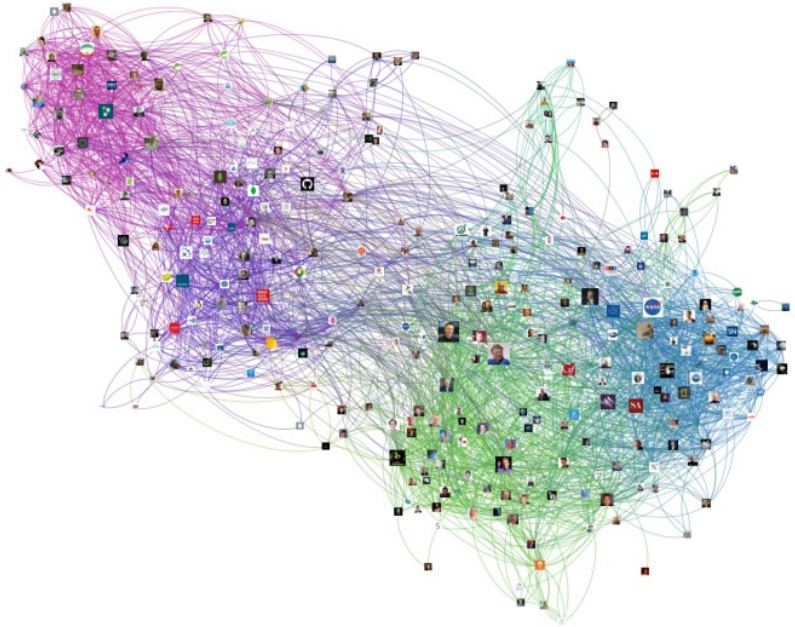
Spring 2021

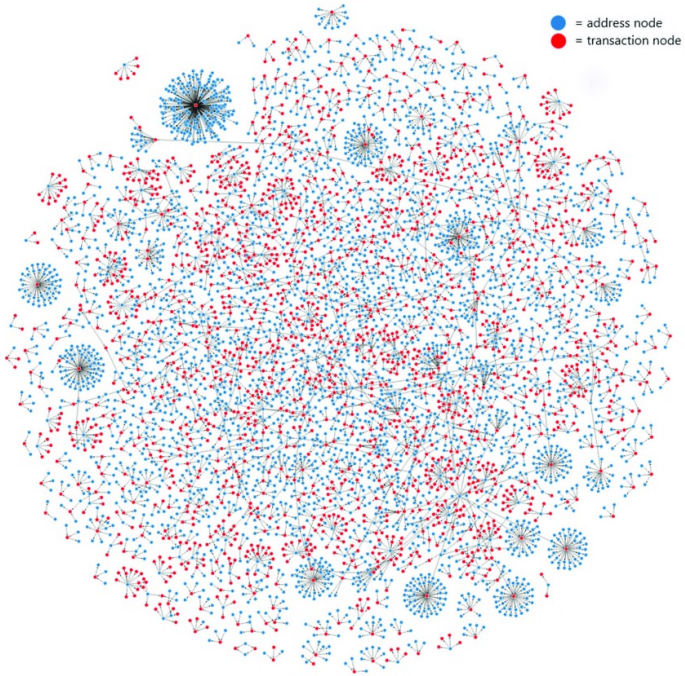
Today start a new unit: algorithms for graphs.

Why graphs? They are a powerful abstraction.

Ex:
- coloring maps
- scheduling constraints
- web graph ($\sim 10^9$ vertices each with 10^2 edges)
- connectome ($\sim 10^{10}$ vertices each with 10^4 edges)
- Bitcoin transaction graph
- social networks
- road maps





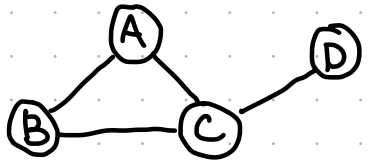


Graph Representations

$n := |V|$, $m := |E|$

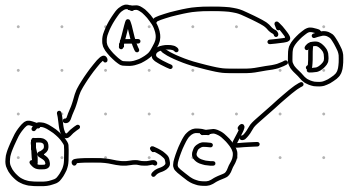
A graph is a pair (V, E) where V is the vertex set and $E \subseteq V \times V$ is the edge set.

- undirected graph:



edge relation is symmetric

- directed graph:



You can represent a graph via:

① adjacency matrix $M_G = \begin{pmatrix} & \vdots^{j} & \\ \cdots & m_{ij} & \cdots \end{pmatrix}^{i} \in \{0,1\}^{n \times n}$

$m_{ij} = 1 \Leftrightarrow (i,j) \in E$

② adjacency list $L_G =$

$1 \mapsto 3, 7, 5, 11$

$3 \mapsto 2, 5, 7$

$\vdots$

Tradeoffs between representations:

	space	$(u,v) \in E$?	neighbors(u)
matrix	$O(\|V\|^2)$	$O(1)$	$O(\|V\|)$
list	$O(\|E\|)$	$O(\text{degree})$	$O(\text{degree})$

Today we study Graph Connectivity. (E.g. which parts of G are reachable from $v \in V$?)

The Explore Procedure

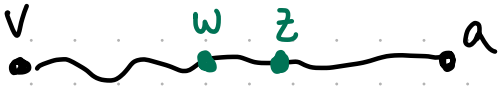
Consider this recursive exploration procedure:

explore (G, v):
- visited[v] := true
- for each $(v, u) \in E$:
  - if not visited[u]: **explore** (G, u)

claim: visits every vertex reachable from v

proof: Suppose there exists $a \in V$ that is reachable from v but is NOT visited.

Look along any path from v to a:

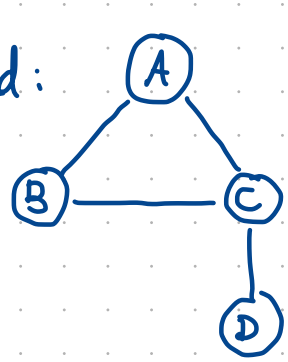


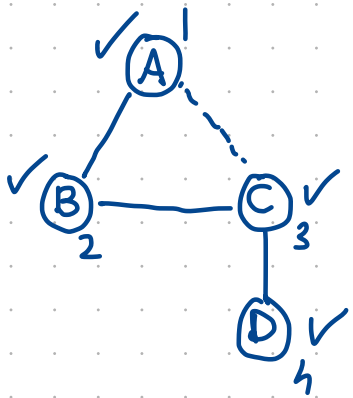
Let z be the first unvisited node, and w the (visited) one before it.

When explore (G, w) was run, it would have run explore (G, z).

This is a contradiction so a was visited. ∎

Ex on undirected:





Depth-First Search (DFS) [undirected case]

To visit the whole graph G we use:

DFS(G): 1. For $v \in V$: visited[v] := false
2. For $v \in V$: if not visited[v] then explore(G, v)

claim: DFS runs in time $O(|V|+|E|)$

Recursion would yield a sloppy bound.

DFS is recursive not due to Divide & Conquer but due to stack.

A more precise accounting is:

- init vector of vertices is $O(|V|)$
- go over each vertex ($O(|V|)$) and maybe explore it

Observation: an undirected edge $(i,j) \in E$ is examined twice [in explore(G,i) and explore(G,j)]

$\Rightarrow O(|E|)$ work in explorations

Connectivity [undirected case]

A graph is connected if every pair of vertices are connected by a path.

The connected components are the subgraphs induced by this equivalence relation.

Q: how to find connected components?

CC(G):

1. For $v \in V$: visited[v] := false
   ccnum[v] := $\perp$
2. cc := 0
3. For $v \in V$:
   if not visited[v]:
   - cc := cc+1
   - explore(G,v)

explore(G,v):

- visited[v] := true
- ccnum[v] := cc
- for each $(v,u) \in E$:
  - if not visited[u]: explore(G,u)

A connected component consists of vertices with the same ccnum value.

(They are the vertices within the same tree in the forest of trees induced by exploration.)

The running time is $O(|V|+|E|)$.

Tracking time in DFS

Maintain global clock & exploration, marking for each vertex the first discovery and final departure.

DFS(G):

1. For $v \in V$: visited[v] := false

   pre[v] := ⊥, post[v] := ⊥

2. clock := 1
3. For $v \in V$:

   if not visited[v] then explore (G,v)

explore (G,v):

- visited[v] := true
- pre[v] := clock
- clock++
- for each $(v,u) \in E$:

  if not visited[u]: explore (G,u)
- post[v] := clock
- clock++

For any vertex $v \in V$: $v \mapsto$ [pre[v], post[v]]

interval on the stack (FILO)

Pre and post numbers determine edge types

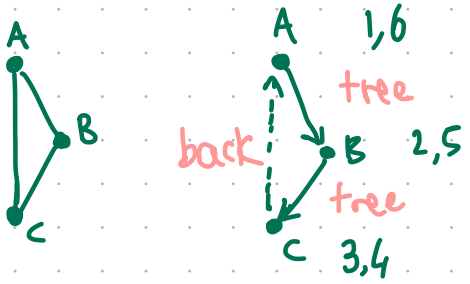
Any two intervals are disjoint or nested (like valid parentheses).

UNDIRECTED CASE

$((u,v) \in E$ iff $(v,u) \in E)$

If $(u,v) \in E$:

- tree edge $[_u \; [_v \;]_v \;]_u$
  part of DFS forest
- back edge $[_v \ldots [_u \;]_u \;]_v$
  to ancestor

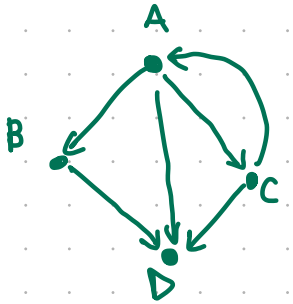


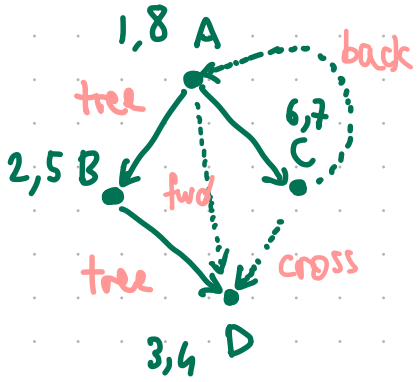
DIRECTED CASE

If $(u,v) \in E$:

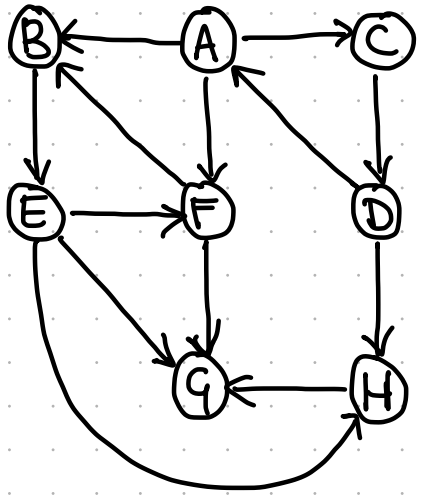
- tree edge $[_u \; [_v \;]_v \;]_u$
  part of DFS forest
- forward edge same as above
  to non-child descendant
- back edge $[_v \; [_u \;]_u \;]_v$
  to ancestor
- cross edge $[_v \;]_v \; [_u \;]_u$
  to already post-visited

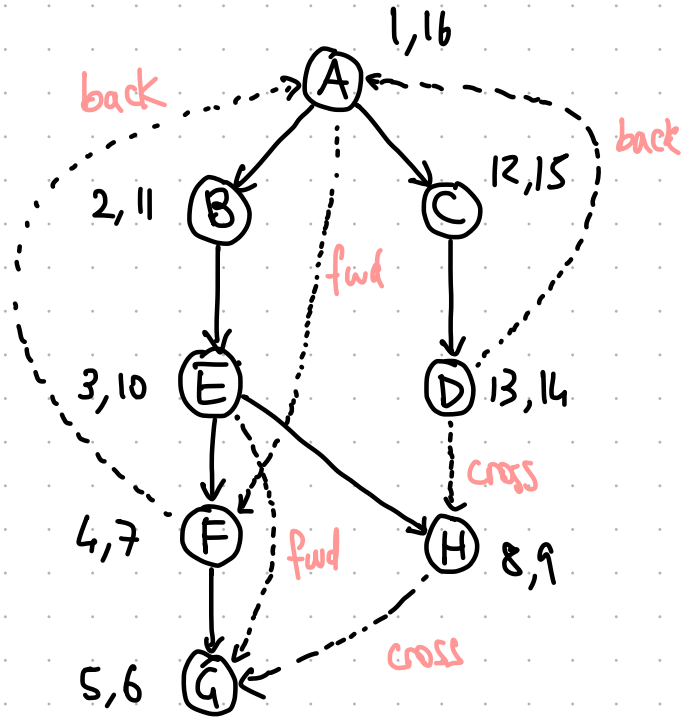
This is impossible: $[_u \;]_u \; [_v \;]_v$





Example on directed graph





What are pre and post numbers useful for?

To begin, we recover an algorithm for finding connected components in undirected graphs:

CC(G) := "Run DFS and then the connected components are according to base level parentheses."

Moreover, we can use them for many other tasks, as we see next (and next lecture).

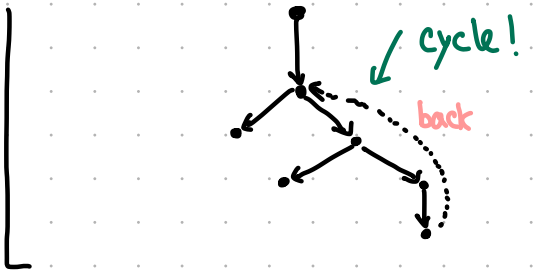
Directed Acyclic Graphs

Here acyclic means without cycles.

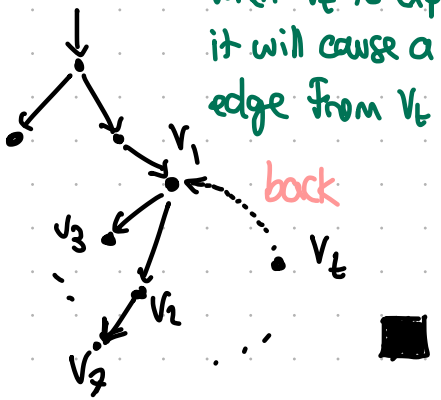
They are useful to model causalities, hierarchies, temporal dependencies, ...

claim: G is acyclic $\Leftrightarrow$ no back edges in DFS(G)

proof: ① back edge → cycle ② cycle → back edge



Say G has cycle $v_1, \dots, v_t$, and WLOG v_1 is visited first when running DFS(G). Then:



The claim directly leads to the following algorithm:

IsDAG(G):
1. Run DFS(G) to collect pre, post numbers.
2. For each $(u,v) \in E$, if (u,v) is a back edge then output NO. } can inline in DFS(G)
   post[v] > post[u]
3. Output YES.

The running time is $O(|V|+|E|)$.