Lecture 2 - Connectivity and Graph Structure

Thursday, January 13, 2022 7:55 AM

Plan for today:

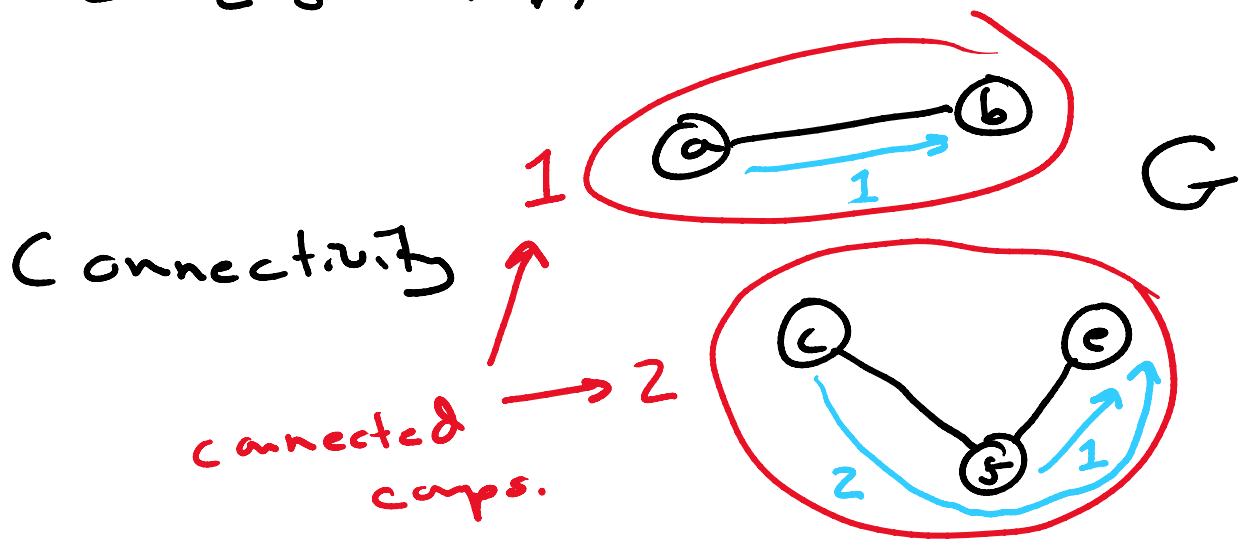
- Review last class
- Biconnectivity and k-connectivity
- Directed graphs, strong and weak connectivity
- The web graph
- Connectivity with networkx
  - Connectivity and weak connectivity
  - Will look at strong next class

Review last class

Graph $G = (V, E)$

$V = \{a, b, c, e, f\}$

$E = \{g = (a, b), h = (c, f), i = (e, f) \ldots\}$



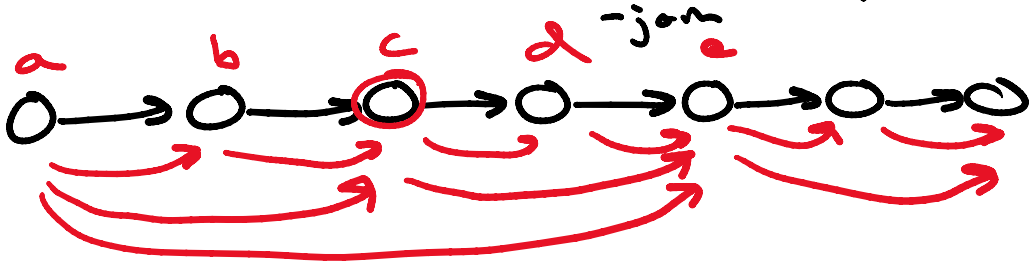
Graph processing

- Can be done using "vertex state"
- We update states for some # iters.

- We update [illegible].
- Problem: Large diameter graph
  ↑ longest shortest path

Solution: pointer-jumping

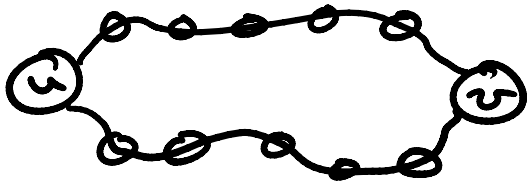
"union-find" approach



↳ only works for certain algorithms

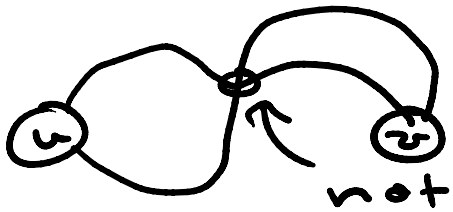
Biconnectivity and k-connectivity

A graph is biconnected if there exists at least 2 vertex-disjoints paths for all u, v vertices in $V(G)$



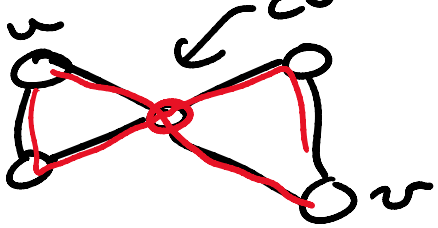
vertex-disjoint

↑ no shared internal vertices



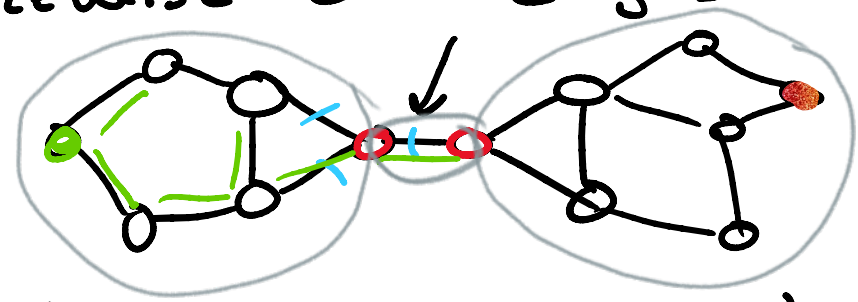
not vertex-disjoint

This implies => no cut vertices



↑ removal of a cut vertex disconnects a graph

Likewise: cut edges

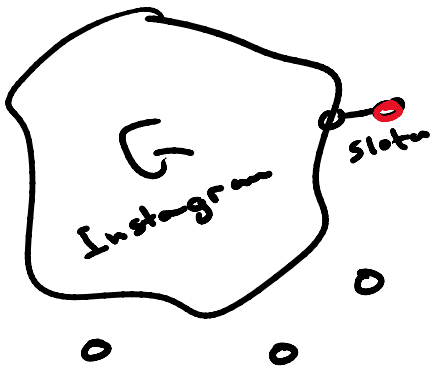


Why do we care in graph mining?

→ Cut vertices, edges are weak or possible failure points in a network

→ Information diffusion gets concentrated at these points

Note: most real-world network are not biconnected (or even connected)



Reason: trivial components are quite common

Trivial component: single disconnected vertex (connectivity)

disconnected vertex (connectivity)

degree-one vertex (biconnectivity)

↳ the neighbor will be a cut vertex

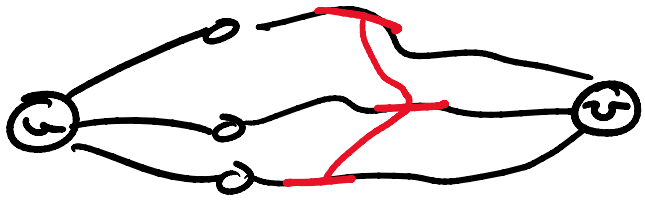
BUT: we might still care about *how* connected a graph is

Generalize all this: k-connectivity

- k: how many vertices we must delete to disconnect a graph
- 1-connected: connectivity
  algorithms: BFS/DFS | label prop.
- 2-connected: biconnectivity
  algorithms: Hopcroft-Tarjan (DFS)
  Slota-Madduri (BFS/label prop)
- 3-connected: triconnectivity
  algorithms: Hopcroft-Tarjan (DFS)
  (k-edge-connectivity)
- k-connected: k-connectivity

- k-connected: k-connectivity

algorithms: network flow



max flow = min cut

Really: this is all relevant to network "robustness"

Usually: we're either asking the question "how many vertices/edges" to remove to disconnect G

OR how many vertices/edges to remove to disconnect v from G

subgraph H from G

Directed graphs

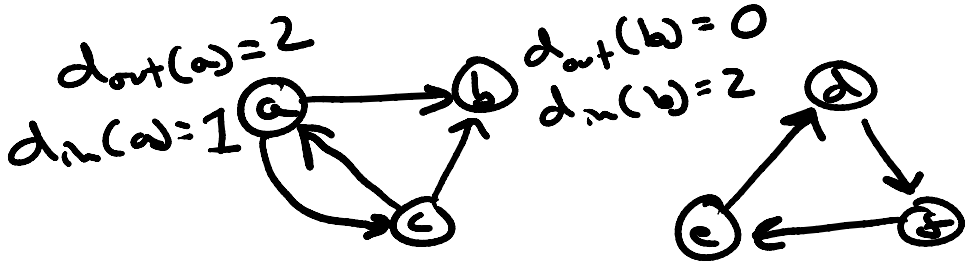
Directed graph $D = (V, E)$

$V = \{a, b, c, e, f\}$

$E = \{g = (a \to b), h = (a \to c) \ldots\}$

$\ldots = 2$ $\quad$ $\ldots(b) = 0$ $\quad$ $i = (c \to a)$

$\vec{e} = (c \to a)$



In undirected graph: $d(v)$ = degree of v (number of attached edges)

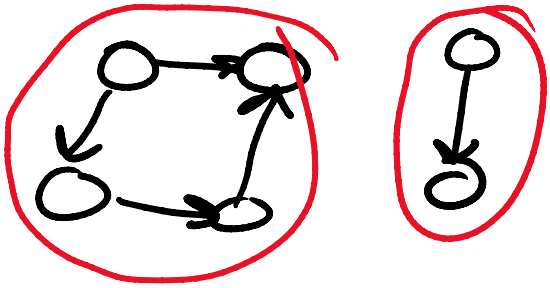
In directed graph:

$d_{in}(v)$ = # incoming edges

$d_{out}(v)$ = # outgoing edges

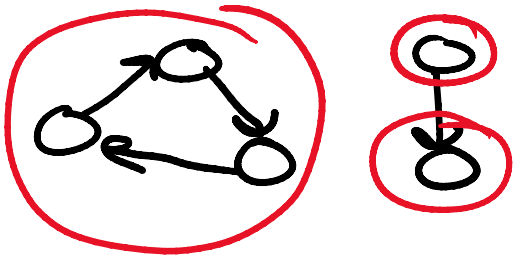
↑ also simply "degree"

Weak connectivity: a graph is weak connected if there exists a u,v-path for all u,v ignoring directionality



Strong connectivity: a graph is strongly

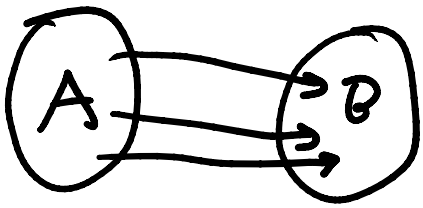
Strong connectivity: a graph is strongly connected if for all u, v pairs there exists a u, v-path



Algorithms: Tarjan (DFS)

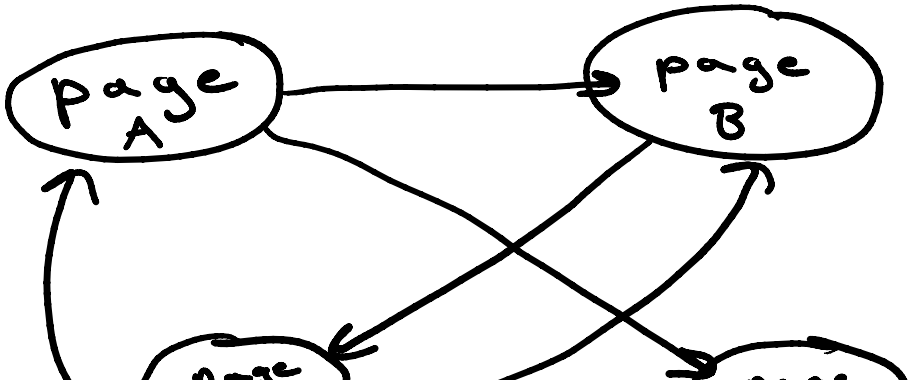
Multistep (Slotactal.) (BFS, color prop.)

Note: the concept of k-connectivity, k-edge-connectivity can also be extrapolated to directed graphs



3-connected

The web graph



Defined on vertices = {web pages}

edges = {hyperlinks}

Structure of the web:

- Not weakly connected
- One "massive" strong component (SCC)
- INs, OUTs, tendrils, tubes

→ bow-tie structure