Lecture 2 - Graph Connectivity

Thursday, January 11, 2024 8:29 AM

Plan for the day:

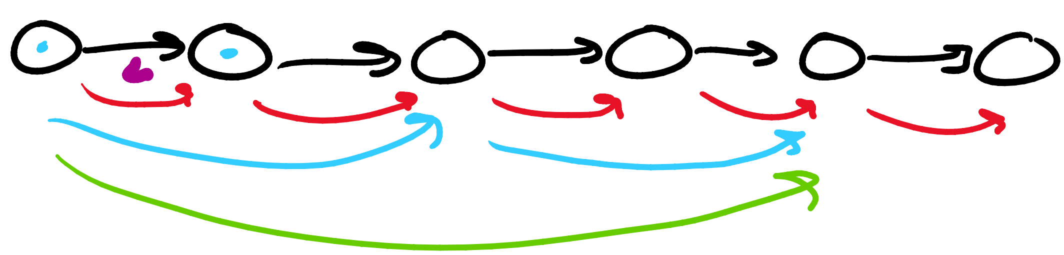
- Last class code example ✓
- Quick review ✓
- Biconnectivity and k-connectivity ✓
- Directed graphs, strong and week connectivity
- The web graph
- Connectivity in NetworkX
  - Connectivity and weak connectivity functions
  - Strong connectivity for next class

Recall our connectivity problem and our algo. solution

→ propagate edge-by-edge

→ upper bound is $\underbrace{\text{diameter}}_{\text{longest shortest path}}$ of the graph

solution: pointer jumping



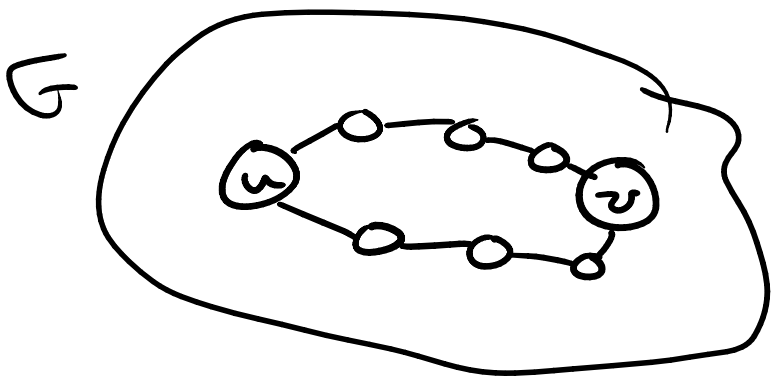
→ note: only works well for certain propagate certain propagative algorithms

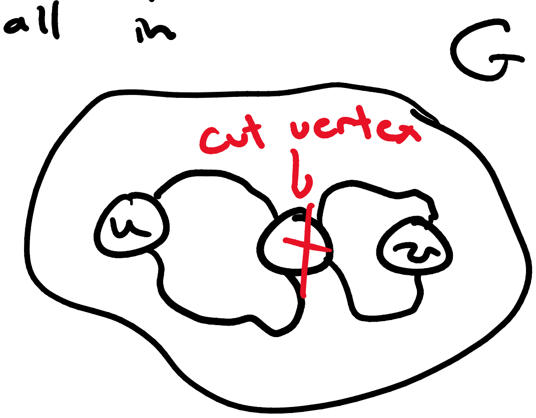
---

Biconnectivity and k-connectivity

↳ a graph is biconnectivity if there exists at least 2 vertex-disjoint paths $\forall u, v \in V(G)$

↑ for all ↑ in

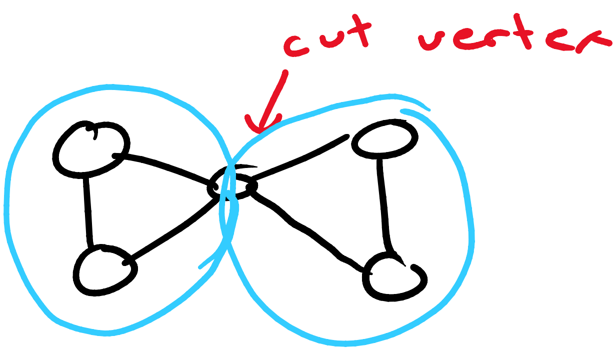




vertex-disjoint paths

not vertex-disjoint

Biconnectivity: there is no cut vertex



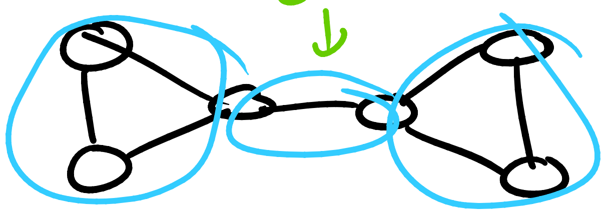
↳ deleting a cut vertex will disconnect a graph

disconnect a graph

biconnected components → maximal biconnected subgraph

biconnectivity decomposition: identifying all biconnected components

cut edge → deleting a cut edge will disconnect the graph



Note: a single edge is technically biconnected

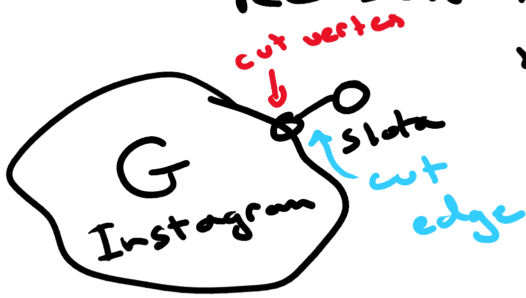
Why do we care?

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

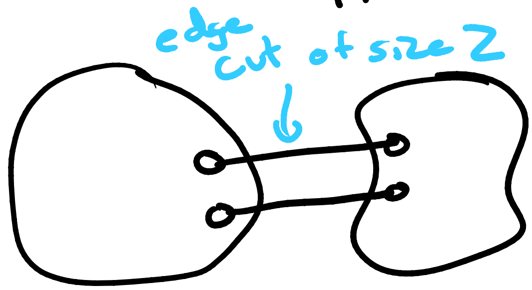
→ Information diffusion gets concentrated at these choke points

Note: most real-world graphs are not biconnected

Reason: trivial components are not uncommon



BUT: we still care about *HOW* connected a graph is



Let's generalize: k-connectivity
k-edge-connectivity

- k: how many vertices/edges we must remove to disconnect a graph

- 1-connected: connectivity

algorithms: BFS/DFS/label prop/pointer jumping
breadth/depth first search

- 2-connected: biconnectivity

- 2-connected: biconnectivity

  algorithms: Tarjan (DFS)

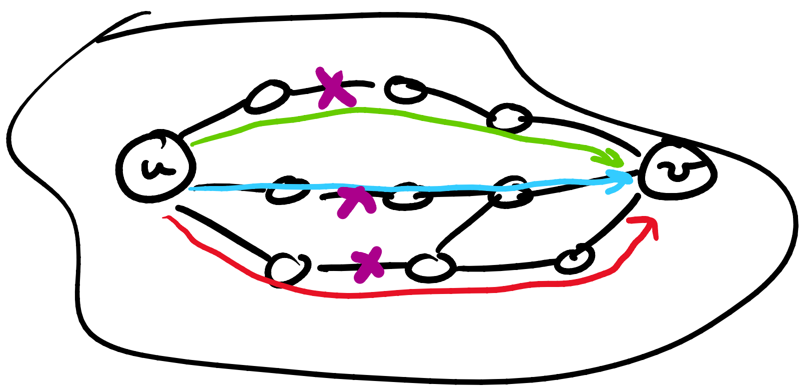
  Slota-Madduri (BFS/label prop)

- 3-connected: triconnectivity

  algorithms: Hopcroft-Tarjan (DFS)

- k-connected: k-connectivity

  algorithms: network flow



max flow = min cut

Really: this is all relevant to network "robustness"

Usually: we asking the question "how many vertices/edges" to disconnect the given network

OR

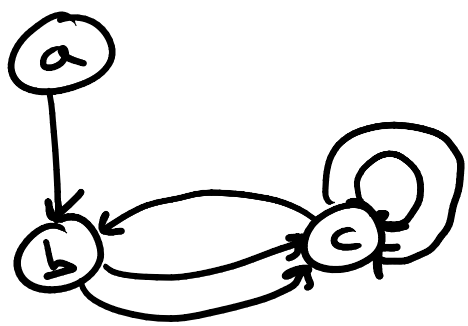
OR
how many to disconnect vertex
u from vertex v

Directed graphs

Directed graph $D = (V, E)$

$V = \{a, b, c, d, \ldots\}$

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



Note, we consider both in-degree d_{in} and out-degree d_{out}

$d_{in}(a) = 0$

$d_{out}(a) = 1$

$d_{in}(b) = 2$

$d_{out}(b) = 2$

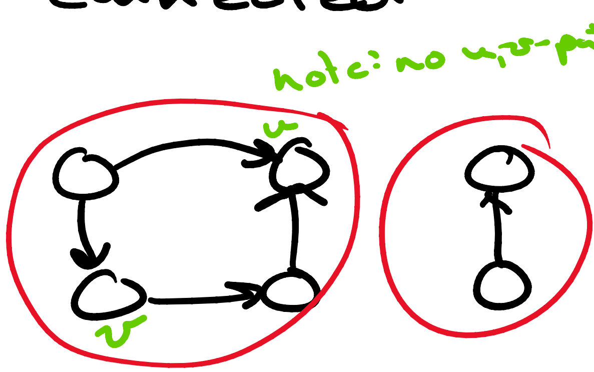
$d_{in}(c) = 4$

$d_{out}(c) = 3$

weak connectivity: a graph is weakly connected if the underlying graph connected if the underlying graph is connected

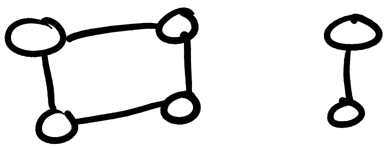
→ the graph if we ignore edge directions

D

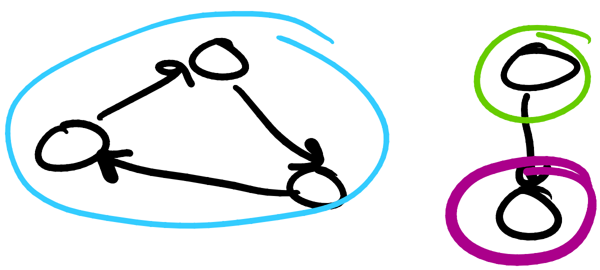


← weak components aka maximal weakly connected subgraphs

underlying graph of D



Strong connectivity: a graph D (directed) is strongly connected if $\forall u,v \in V(D)$: $\exists$ u,v-path



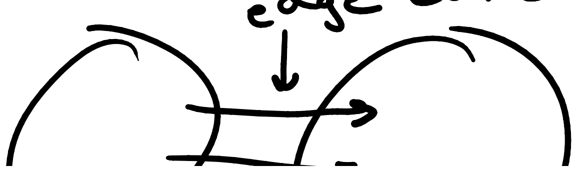
D has 3 strong components

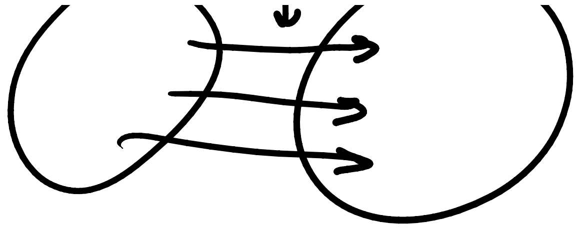
Algorithms: Tarjan again (DFS)

Multistep (Slota et. al)

Note: k-connectivity and edge connectivity can also be generalized to directed graphs

edge cut of size 3

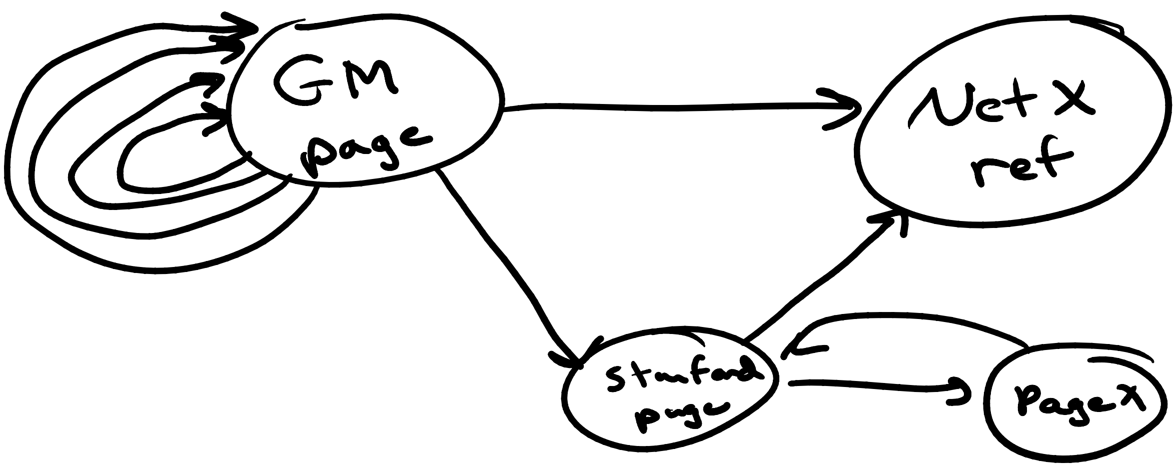




Let's put it to practice via the web graph

vertices: web pages

edges: links between pages

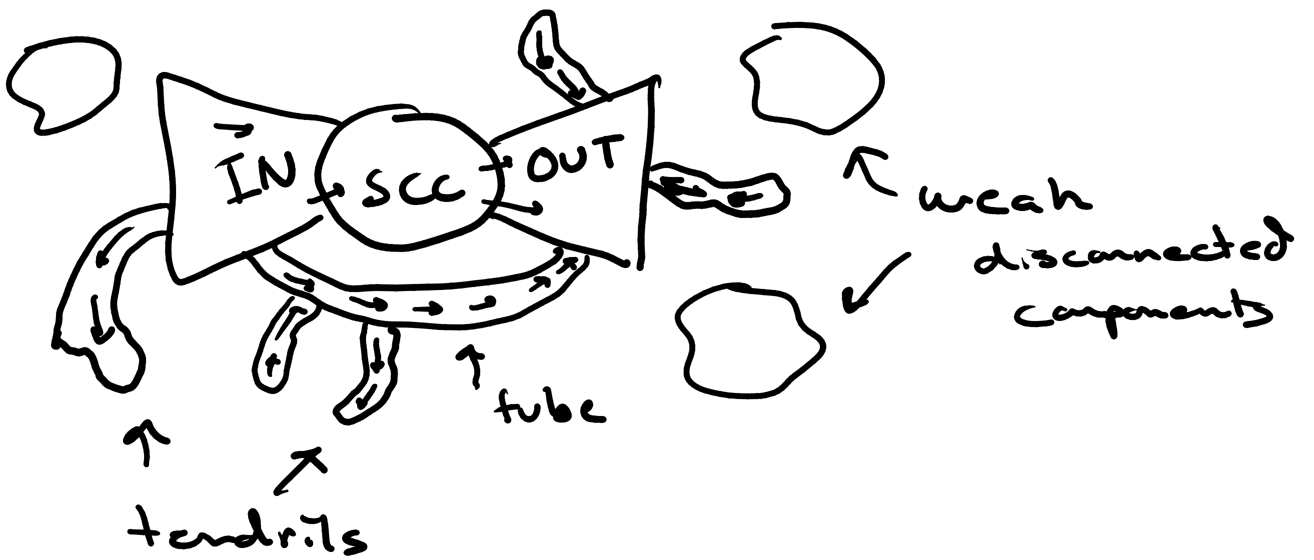


The web graph consider all $\underbrace{\text{pages}}_{\text{vertices}}$ and all $\underbrace{\text{links}}_{\text{edges}}$

The structure of the web

- Not weakly connected
- One "massive" strong/weak component
- Defined IN, OUT, tendril, tubes
  - IN: can reach SCC (strong component)
  - OUT: can be reached by SCC
  - tendril: everything else
  - tubes: can reach OUT, can be reached by IN

→ Bow-tie structure



To determine set membership:

weak component → easy decomposition

SCC → strong connectivity decomposition

decomposition

OUT → traverse from an SCC vert

IN → traverse backwards from an SCC vert