

# Announcements



HW2

- Out tonight
- Due Mon 2/28

Assignment schedule

- Draft deadlines on course website

# Plan

Graphs

Start on text processing if time

# 15-388/688 - Practical Data Science: Graph and network processing

Pat Virtue  
Carnegie Mellon University  
Spring 2022

# Outline

Networks and graph

Representing graphs

Graph algorithms

Graph libraries

# Outline

Networks and graph

Representing graphs

Graph algorithms

Graph libraries

# Networks vs. graphs?

Our terminology (fairly standard, though some use them differently):

Networks are the systems of interrelated objects (in the real world)

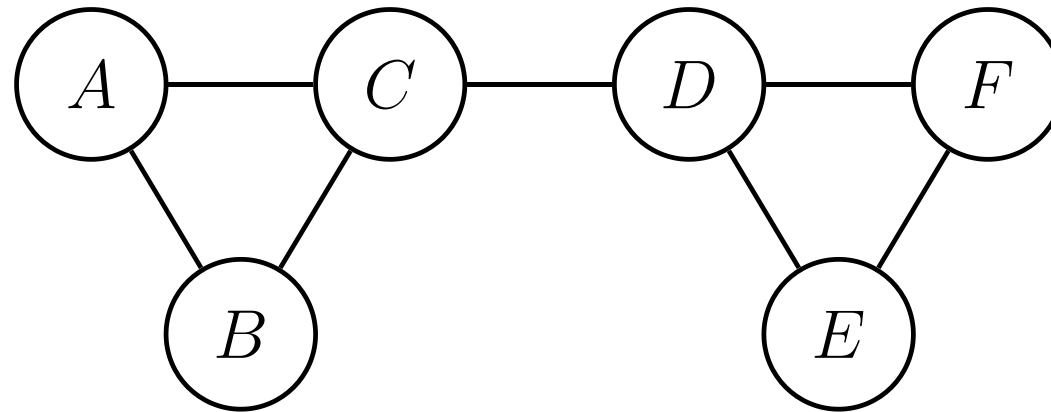
Graphs are the mathematical model for representing networks

This lecture is largely about representations and algorithms for graphs

But of course, in data science we use these algorithms to answer questions about networks

# Graphs models

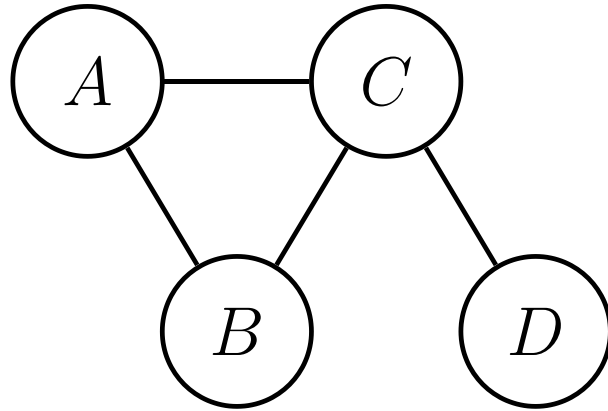
A graph is a collection of vertices (nodes) and edges  $\underline{G} = (V, E)$



$$V = \{A, B, C, D, E, F\}$$

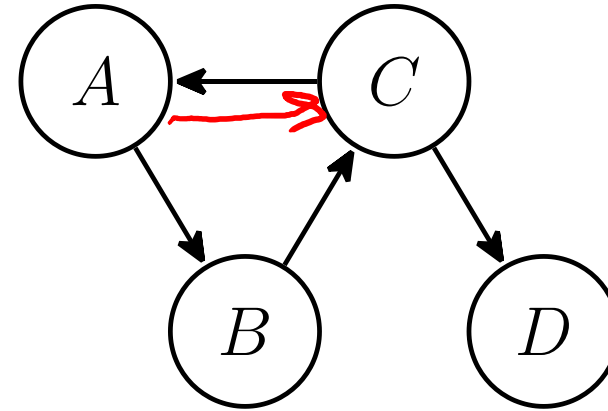
$$E = \{(A, B), (A, C), (B, C), (C, D), (D, E), (D, F), (E, F)\}$$

# Directed vs. undirected graphs



**Undirected**

E.g. paper co-authorship



**Directed**

E.g. web links

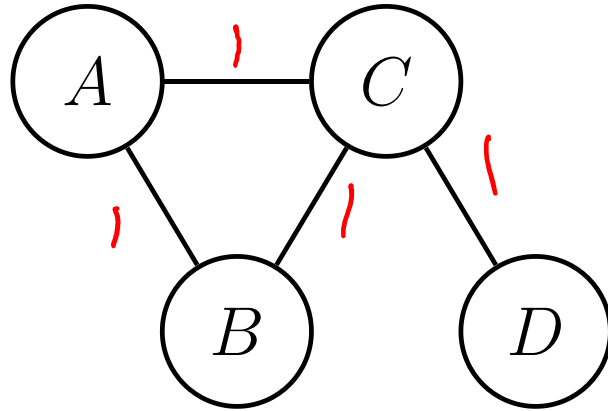


# Poll 1

For a network of roads in a city/state, should the graph be:

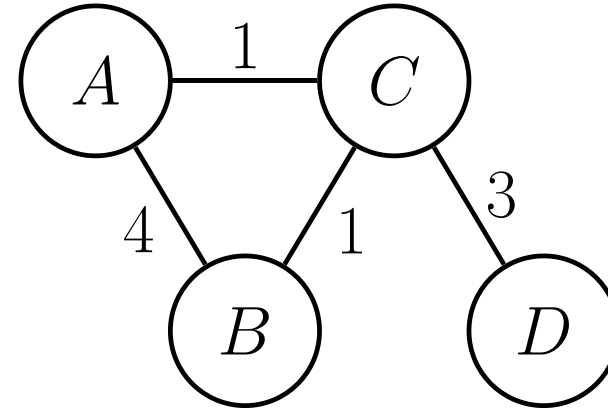
- A. Undirected
- B. Directed

# Weighted vs. unweighted graphs



## **Unweighted**

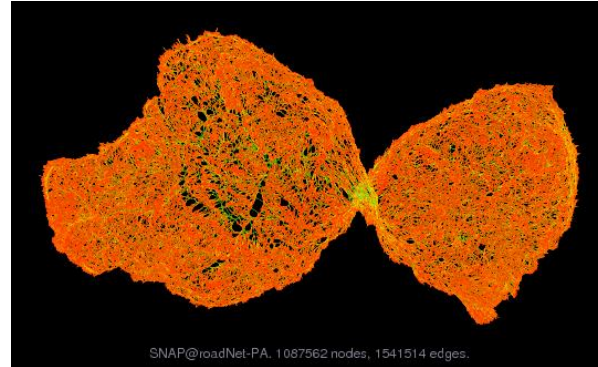
E.g. friends on social network



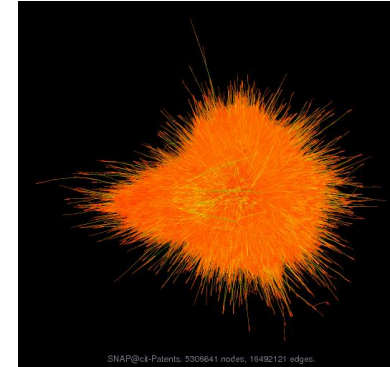
## **Weighted**

E.g. travel distance between cities

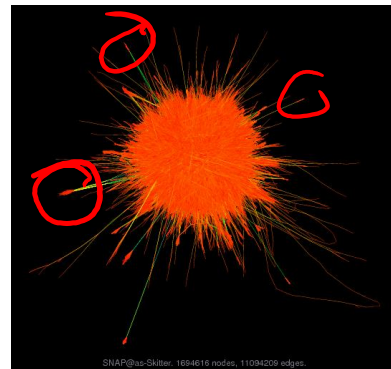
# Some example graphs



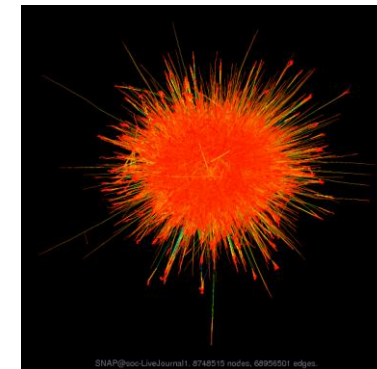
PA road network:  
1M nodes, 3M edges



Patent citations:  
3.7M nodes, 16.5M edges



Internet topology (in 2005)  
1.6M nodes, 11M edges



LiveJournal social network  
4.8M nodes, 69M edges



# Outline

Networks and graph

Representing graphs

Graph algorithms


Graph libraries

# Representations of graphs

There are a few different ways that graphs can be represented in a program, which one you choose depends on your use case

E.g., are you going to be modifying the graph dynamically (adding/removing nodes/edges), just analyzing a static graph, etc?

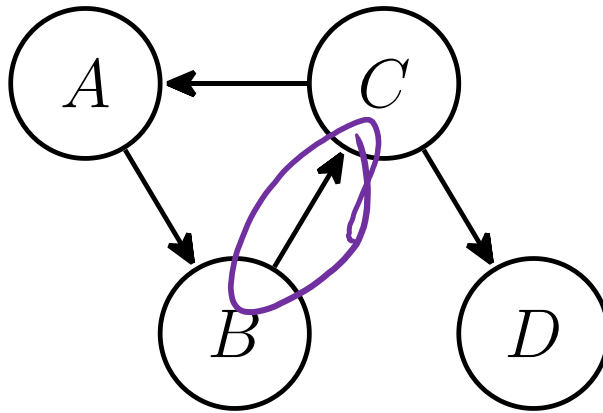
Three main types we will consider:

- 1. Adjacency list
- ✓ 2. Adjacency dictionary
- ✓ 3. Adjacency matrix 

# Adjacency list

order not sorted

For each node, store an array of the nodes that it connects to



Node	Edges
A	[B]
B	[C]
C	[A,D]
D	[]

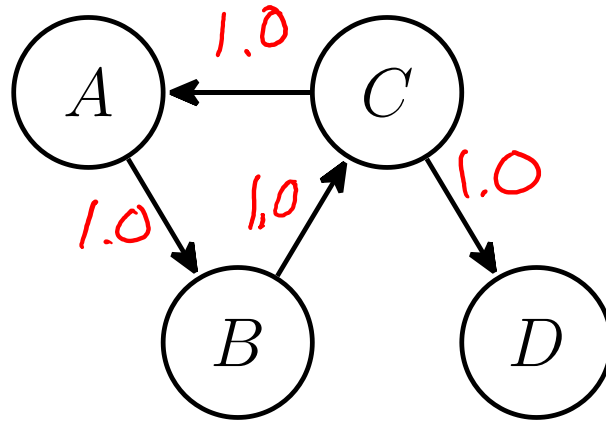
List  
Dict  
↓  
A [B]  
B [C]  
C [D, A]  
D []

**Pros:** easy to get all outgoing links from a given node, fast to add new edges (without checking for duplicates)

**Cons:** deleting edges or checking existence of an edge requires scan through given node's full adjacency array

# Adjacency dictionary

For each node, store a dictionary of the nodes that it connects to



*dict*

Node (key)	Edges
A	{B:1.0}
B	{C:1.0}
C	{A:1.0,D:1.0}
D	{}

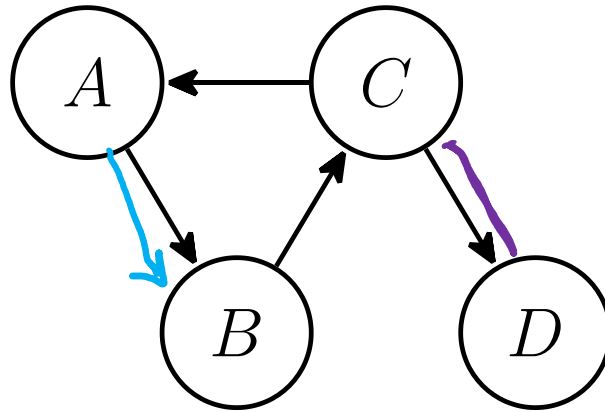
*Dict (Set, unweighted)*

**Pros:** easy to add/remove/query edges (requires two dictionary lookups, so a  $O(1)$  operation)

**Cons:** overhead of using a dictionary over array

# Adjacency matrix

Store the connectivity of the graph as a matrix



$$A = \begin{matrix} & \begin{matrix} \text{(From)} \\ A & B & C & D \end{matrix} \\ \begin{matrix} A \\ B \\ C \\ D \end{matrix} \text{ (To)} & \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \end{matrix}$$

In virtually all cases, you will want to store this as a sparse matrix

Pros/cons depend on which sparse matrix format you use, but most operations on a static graph will be much faster using the right format

✧ Connection between adjacency list and sparse CSC format



# Outline

Networks and graph

Representing graphs

Graph algorithms

Graph libraries

# Graph algorithms

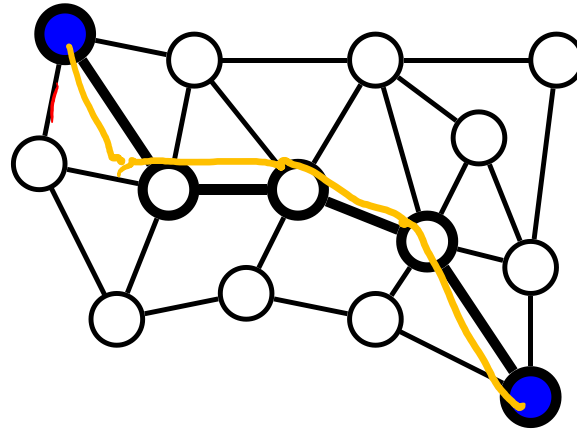
Algorithms for graphs could be (in fact, is) an entire course on its own

We're going to briefly highlight just three algorithms that address different problem classes in graphs

1. Finding shortest paths in a graph – Dijkstra's algorithm
2. Finding important nodes in a graph – PageRank
3. Finding communities in a graph – Girvan-Newman

# Shortest path problem

Classical graph problem: find the shortest path between two nodes



Some important distinctions or modifications

Weighted vs. unweighted, directed vs. undirected, negative weights

→ Single-source shortest path (we'll do this one)

All-pairs shortest path

# Dijkstra's algorithm

Algorithm for single-source shortest path

**Basic idea:** dynamic programming algorithm, at each node maintain an *upper bound* on distance to source, iteratively expand node with smallest upper bound (updating bounds of its neighbors)

**Given:** Graph  $G = (V, E)$ , Source  $s$

**Initialize:**

$$D[s] \leftarrow 0, D[i \neq s] \leftarrow \infty$$
$$Q \leftarrow V$$

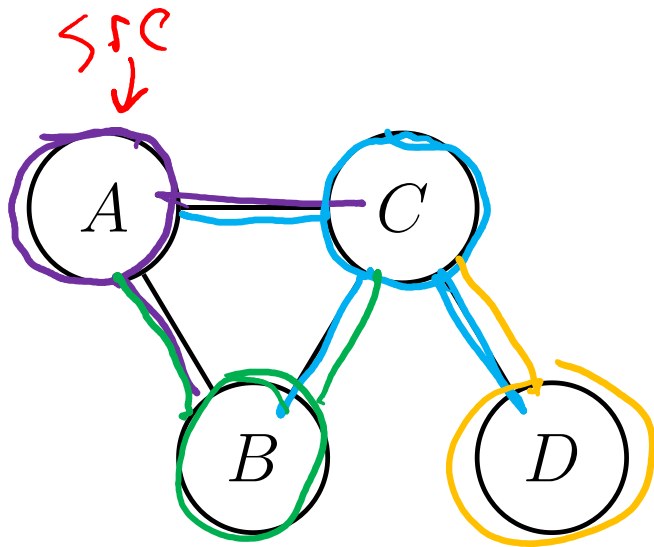
**Repeat** until  $Q$  empty:

$i \leftarrow$  Remove element from  $Q$  with smallest  $D$

For all  $j$  such that  $(i, j) \in E$ :

$$D[j] = \min(D[j], D[i] + 1)$$

# Dijkstra's algorithm example



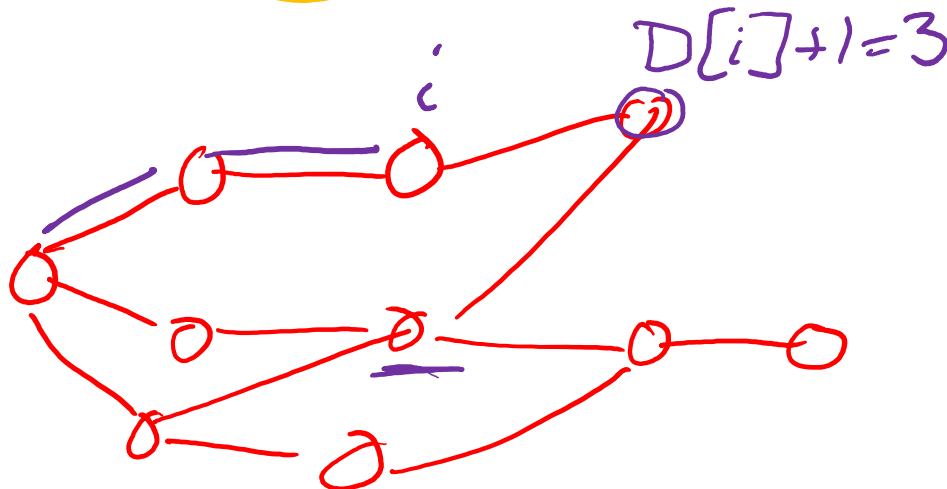
pop  $Q \rightarrow i$

$D[i]$

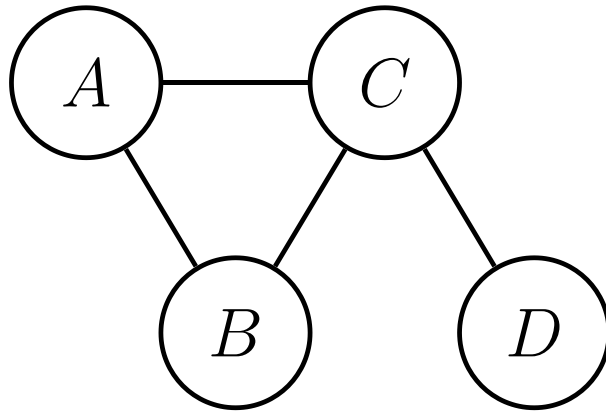
$child = D[i] + 1$

Q	D
<del>A</del>	A [0] $\leftarrow x$
<del>B</del>	B [1] $\leftarrow x$
<del>C</del>	C [1]
<del>D</del>	D [2] $\leftarrow \checkmark$

$D[i] = 2$



# Dijkstra's algorithm example



Initialization: source  $A$

$$D = [0, \infty, \infty, \infty]$$

$$Q = \{A, B, C, D\}$$

Step 1: Pop node  $A$

$$Q = \{B, C, D\}$$

$$D = [0, 1, 1, \infty]$$

Step 2: Pop node  $B$

$$Q = \{C, D\}$$

$$D = [0, 1, 1, \infty]$$

Step 3: Pop node  $C$

$$Q = \{D\}$$

$$D = [0, 1, 1, 2]$$

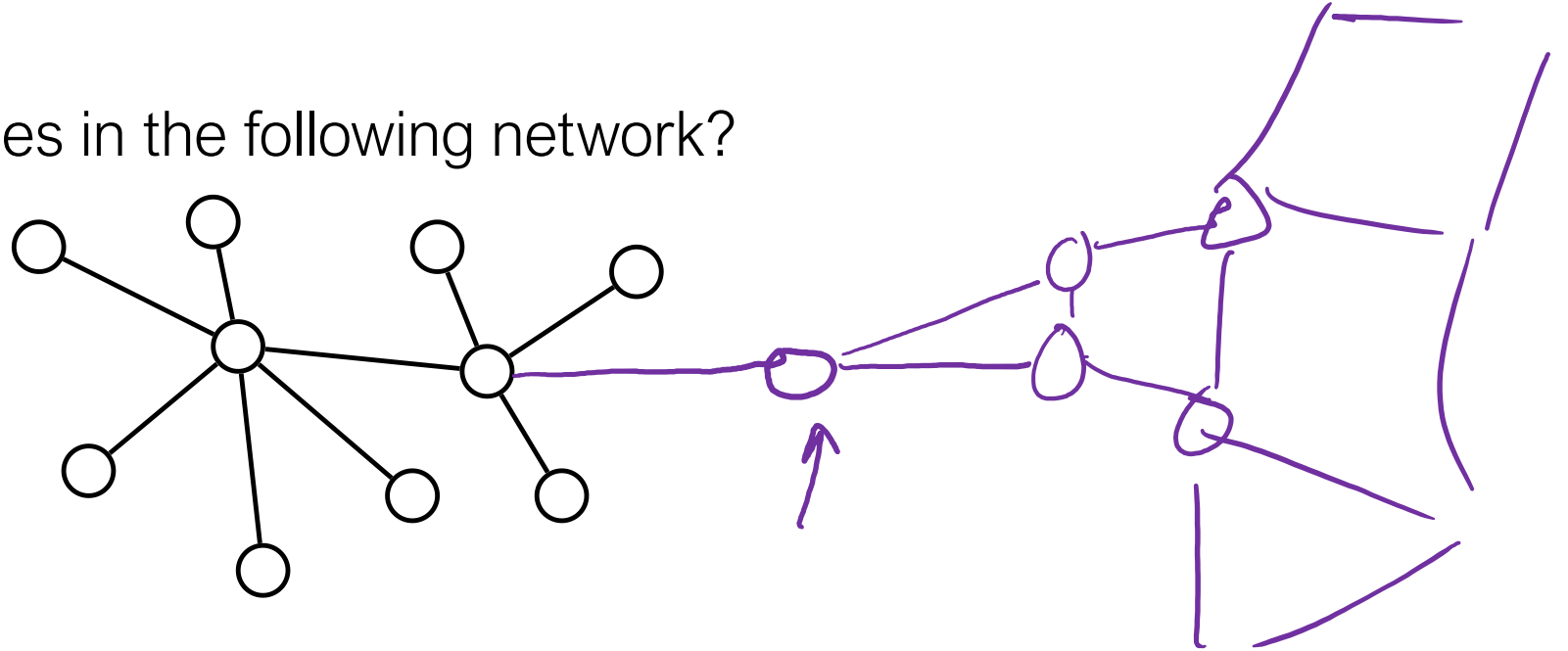
Step 4: Pop node  $D$

$$Q = \{\}$$

$$D = [0, 1, 1, 2]$$

# “Important” nodes

What are the important nodes in the following network?



Unlike shortest path, there is not correct answer here, depends on how you define importance

# PageRank algorithm

The algorithm that started Google

Perspective on importance: consider a *random walk* on the graph

We start at a random node

We repeatedly jump to a random neighboring node

If the node has no outgoing edges (in directed graph), jump to a random node  
(Optionally) also jump to a random node with probability  $(1 - d)$

*Note:* In other words,  $d$  is the probability of continuing, i.e., the damping factor

Node importance is the probability that we will be at a given node when following the above procedure



# PageRank algorithm

**Given:** Graph  $G = (V, E)$ , restart probability  $(1 - d)$ , iteration count  $T$

Let  $\hat{P}$  be an  $|V| \times |V|$  matrix where  $\hat{P}_{i,j}$  is the probability of transitioning from node  $j$  to node  $i$  for one step in the random walk algorithm.

Let  $x$  be a length  $|V|$  vector where  $x_k$  is the probability of being at node  $k$  at some time step.

**Repeat**  $T$  times:

$$x \leftarrow \hat{P}x$$

For those who have heard these terms, this algorithm is creating a Markov chain over the graph, and finding the stationary distribution (largest eigenvector) of this Markov chain

# PageRank algorithm

$$\hat{x} = \hat{P} x$$

$N$

$\begin{bmatrix} .2 \\ .3 \\ .1 \\ .4 \end{bmatrix} = \begin{bmatrix} & & & \\ & \square & & \\ & & & \\ & & & \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$

A  
B  
C  
D

$\hat{P}_{ij}$

prob of transitioning from  
node  $j \rightarrow$  node  $i$ .

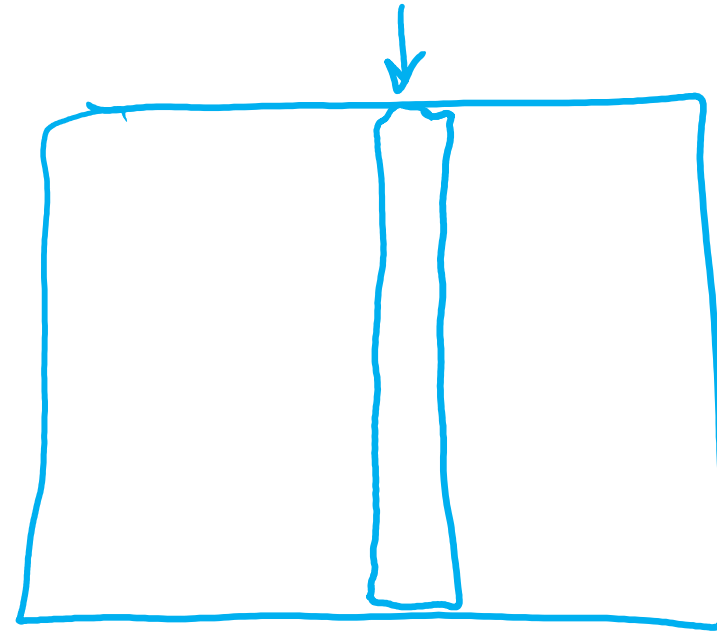
$x_k$

prob of being node  $k$  at some  $t$

## Poll 2

$\hat{P}_{ij}$  is the probability of transitioning from node  $j$  to node  $i$  in a random walk. Which of the following must be true

- ✓ A.  $\sum_{i=1}^N \hat{P}_{i,j} = 1 \quad \forall j$
- B.  $\sum_{j=1}^N \hat{P}_{i,j} = 1 \quad \forall i$
- ~~C.~~ C.  $\sum_{i=1}^N \sum_{j=1}^N \hat{P}_{i,j} = 1 = N$

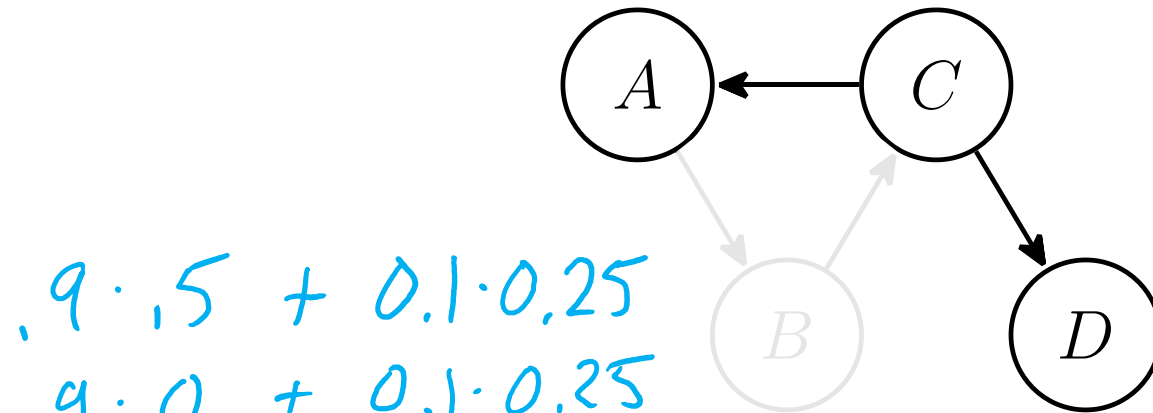


# PageRank algorithm

**Given:** Graph  $G = (V, E)$ ,  
restart probability  $(1 - d)$

Let  $\hat{P}$  be an  $|V| \times |V|$  matrix  
where  $\hat{P}_{i,j}$  is the probability of  
transitioning from node  $j$  to  
node  $i$  for one step in the  
random walk algorithm.

Let  $x$  be a length  $|V|$  vector  
where  $x_k$  is the probability of  
being at node  $k$  at some time  
step.



$$\begin{array}{c}
 \begin{bmatrix} .475 \\ .025 \\ .025 \\ .475 \end{bmatrix} \leftarrow \begin{array}{c} A \\ B \\ C \\ D \end{array} \begin{bmatrix} & & 0.475 \\ & & 0.025 \\ & & 0.025 \\ & & 0.475 \end{bmatrix} \begin{array}{c} A \\ B \\ C \\ D \end{array} \begin{bmatrix} 0 \\ 0 \\ 1.0 \\ 0 \end{bmatrix}
 \end{array}$$

$\hat{P}$

.025

# PageRank algorithm

**Given:** Graph  $G = (V, E)$ , restart probability  $(1 - d)$ , iteration count  $T$

**Initialize:**

$A \leftarrow \text{Adjacency\_Matrix}(G)$

$P \leftarrow$  replace zero columns of  $A$  with  $\mathbf{1}$ , and normalize columns

$\hat{P} \leftarrow dP + (1 - d) \frac{1}{|V|} \mathbf{1}\mathbf{1}^T$

$x \leftarrow \frac{1}{|V|} \mathbf{1}$

**Repeat**  $T$  times:

$x \leftarrow \hat{P}x$

$$\mathbf{1}\mathbf{1}^T = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & \dots & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & \dots & 1 \\ 1 & 1 & \dots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \dots & 1 \end{bmatrix}$$

For those who have heard these terms, this algorithm is creating a Markov chain over the graph, and finding the stationary distribution (largest eigenvector) of this Markov chain

# PageRank example

$A \leftarrow \text{Adjacency\_Matrix}(G)$

$P \leftarrow$  replace zero columns of  $A$  with  $\mathbf{1}$ , and normalize columns

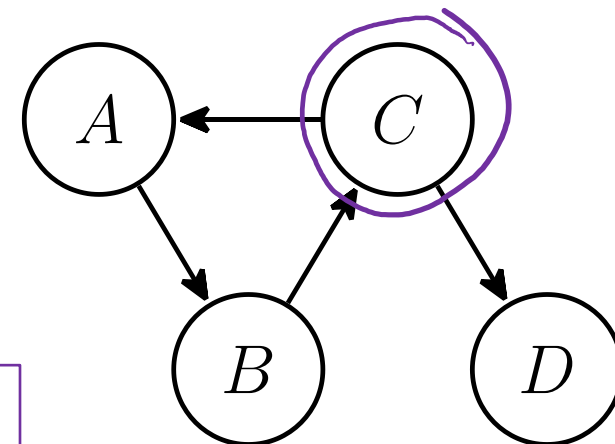
$$\hat{P} \leftarrow dP + (1 - d) \frac{1}{|V|} \mathbf{1}\mathbf{1}^T$$

$d = 0.9$

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

$$P = \begin{bmatrix} 0 & 0 & 0.5 & 0.25 \\ 1 & 0 & 0 & 0.25 \\ 0 & 1 & 0 & 0.25 \\ 0 & 0 & 0.5 & 0.25 \end{bmatrix}$$

$$\hat{P} = 0.9 \begin{bmatrix} 0 & 0 & 0.5 & 0.25 \\ 1 & 0 & 0 & 0.25 \\ 0 & 1 & 0 & 0.25 \\ 0 & 0 & 0.5 & 0.25 \end{bmatrix} + 0.1 \begin{bmatrix} 0.25 & 0.25 & 0.25 & 0.25 \\ 0.25 & 0.25 & 0.25 & 0.25 \\ 0.25 & 0.25 & 0.25 & 0.25 \\ 0.25 & 0.25 & 0.25 & 0.25 \end{bmatrix} = \begin{bmatrix} 0.025 & 0.025 & 0.475 & 0.25 \\ 0.925 & 0.025 & 0.025 & 0.25 \\ 0.025 & 0.925 & 0.025 & 0.25 \\ 0.025 & 0.025 & 0.475 & 0.25 \end{bmatrix}$$



Initialize:  $x \leftarrow \frac{1}{|V|} \mathbf{1}$

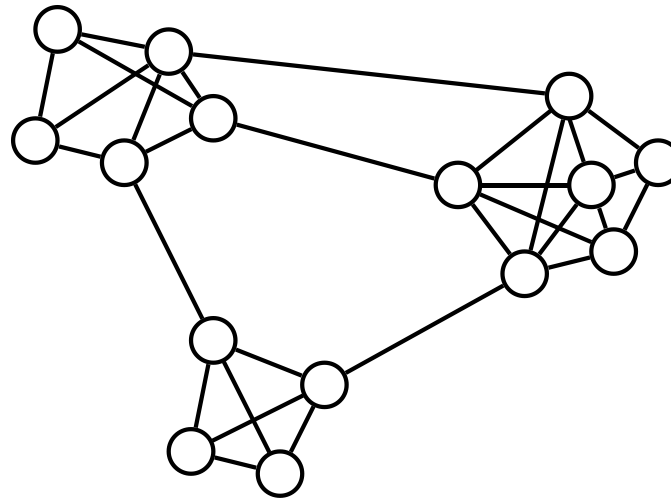
Repeat  $T$  times:

$$x \leftarrow \hat{P}x$$

$$x = \begin{bmatrix} 0.25 \\ 0.25 \\ 0.25 \\ 0.25 \end{bmatrix} \rightarrow \rightarrow \rightarrow \rightarrow \begin{bmatrix} 0.21 \\ 0.26 \\ 0.31 \\ 0.21 \end{bmatrix}$$

# Community detection

**Community:** subgraphs where nodes are densely connected to each other, but sparsely connected to other nodes



A “soft” version of a clique (a fully connected subgraph)

A fundamental concept in e.g. social networks

# Girvan-Newman Algorithm

Published in 2002 (Girvan and Newman, 2002), one of the first methods of “modern” community detection

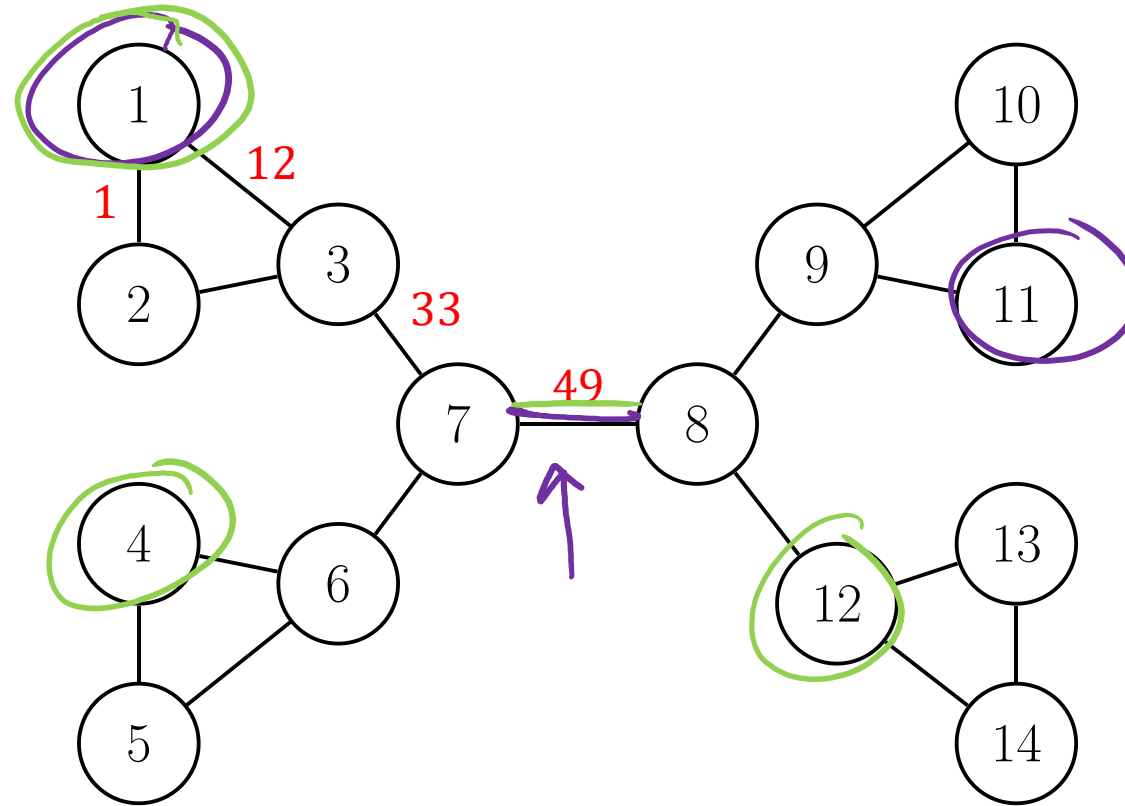
**Basic idea:** Recursively partition the network by removing edges, groups that are last to be partitioned are “communities”

1. Compute “betweenness” of edges in the network = number of shortest paths that pass through each edge
2. Remove edge(s) with highest betweenness, if this breaks the graph into subgraphs, recursively partition each one
3. Result is a hierarchical partitioning of the graph

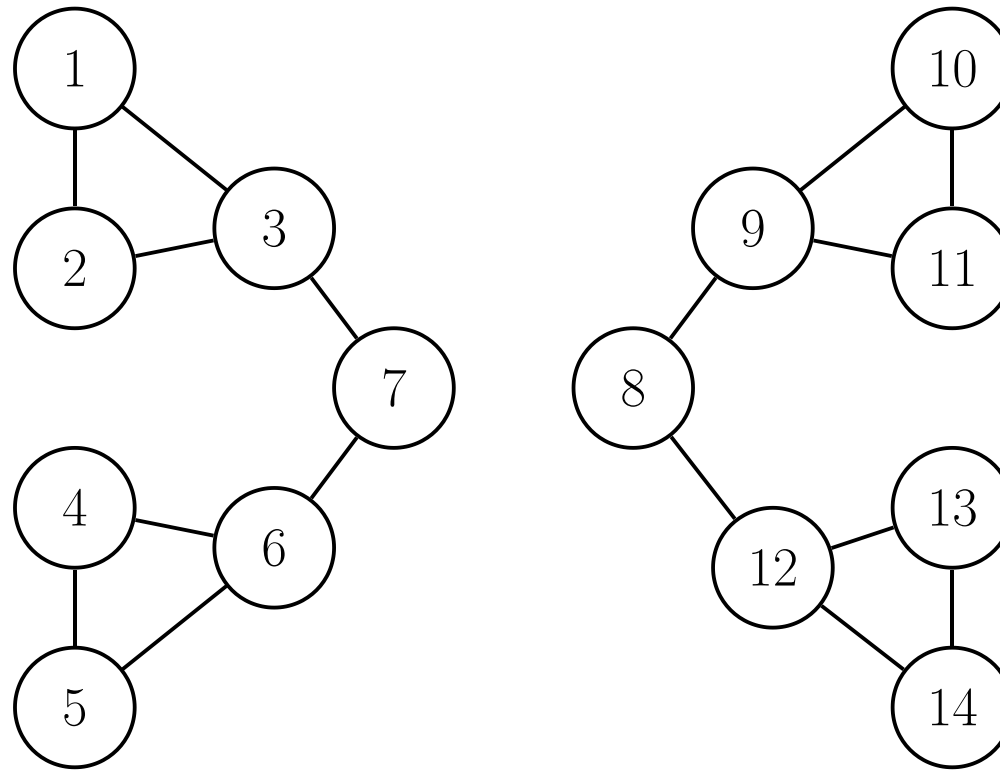
Challenge is efficiently computing betweenness as we partition graph (we will not cover this)



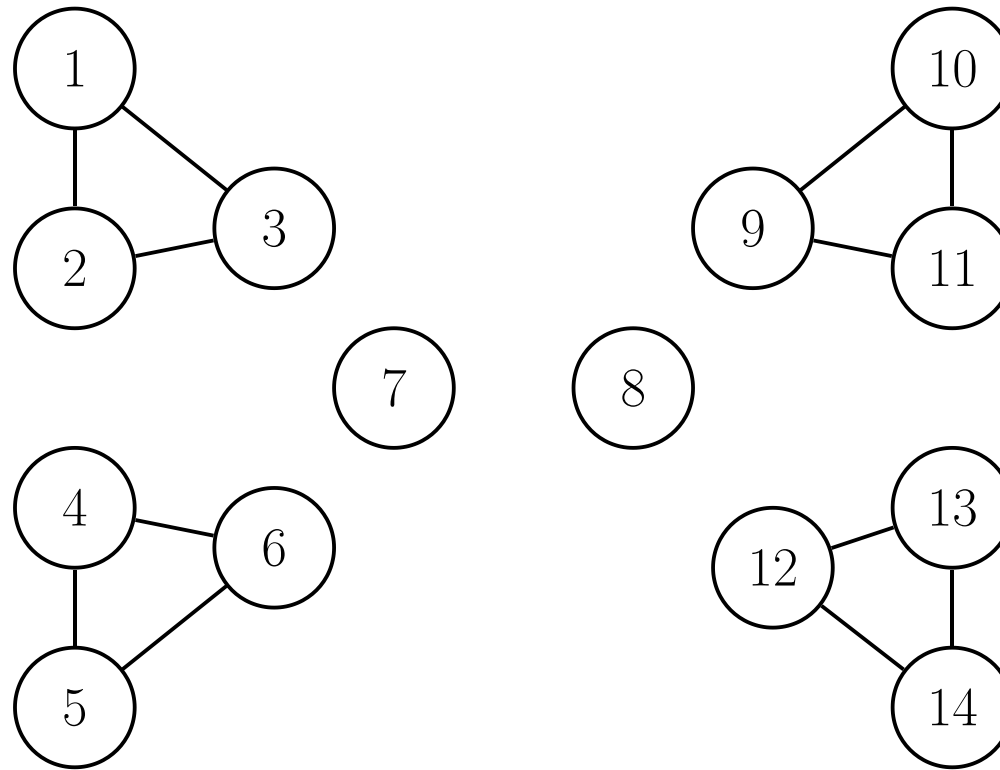
# Algorithmic illustration



# Algorithmic illustration



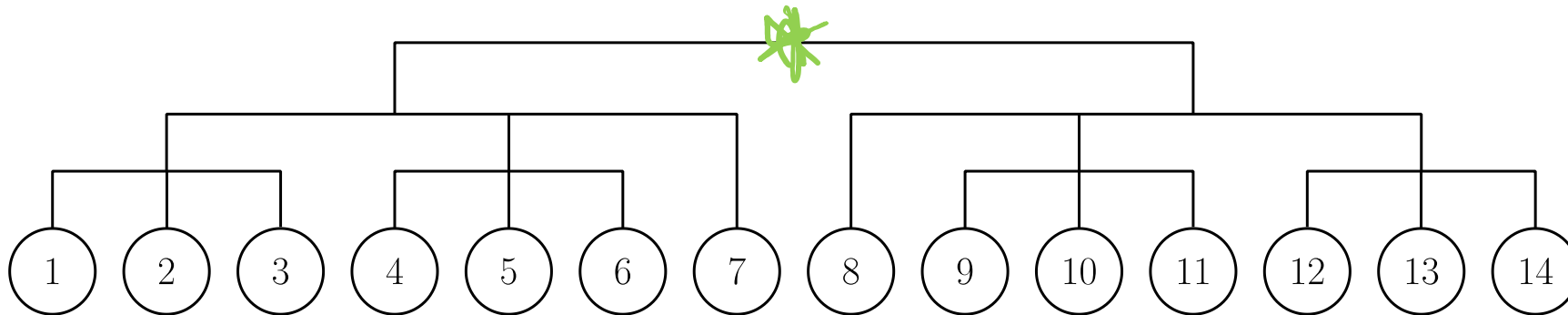
# Algorithmic illustration



# Resulting hierarchy (dendrogram)

Communities can be extracted by looking at the grouping at different levels of the tree

May want to threshold on things like community size, etc



# Outline

Networks and graph

Representing graphs

Graph algorithms

Graph libraries

# NetworkX

NetworkX: Python library for dealing with (medium-sized) graphs

<https://networkx.github.io/>

Simple Python interface for constructing graph, querying information about the graph, and running a large suite of algorithms

*Not* suitable for very large graphs (all native Python, using adjacency dictionary representation)

# Creating graphs

Create an undirected or directed graph

```
import networkx as nx

G = nx.Graph()    # undirected graph
G = nx.DiGraph()  # directed graph
```

Add and remove nodes/edges

```
# add and remove edges
G.add_edges_from([("A", "B"), ("B", "C"), ("C", "A"), ("C", "D")])
G.remove_edge("A", "B")
G.add_edge("A", "B")
G.remove_edges_from([("A", "B"), ("B", "C")])
G.add_edges_from([("A", "B"), ("B", "C")])
# also add_node(), remove_node(), add_nodes_from(), remove_nodes_from()
```

# Nodes/edges and properties

NetworkX uses adjacency dictionary format internally

```
print G["C"]  
# {'A': {}, 'D': {}}
```

Iterate over nodes and edges

```
for i in G.nodes():    # loop over nodes  
    print i  
for i,j in G.edges(): # loop over edges  
    print i,j
```

Get and set node/edge properties

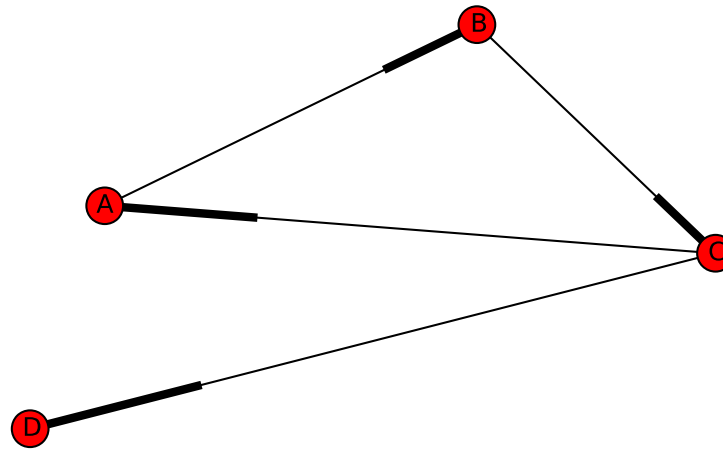
```
G.node["A"]["node_property"] = "node_value"  
G.edge["A"]["B"]["edge_property"] = "edge_value"  
G.nodes(data=True) # iterator over nodes returning properties  
G.edges(data=True) # iterator over edges returning properties
```



# Drawing and node properties

Draw a graph using matplotlib (not the best visualization)

```
import matplotlib.pyplot as plt
%matplotlib inline
nx.draw(G, with_labels=True)
plt.savefig("mpl_graph.pdf")
```



# Algorithms

Almost all the (medium scale) algorithms you could want

```
nx.shortest_path_length(G, source="A") # iterater over path lengths

nx.pagerank(G, alpha=0.9) # dictionary of node ranks

# NOTE: this requires Networkx 2.0
nx.girvan_newman(G) # iterator over partitions at  
                    different hierarchy levels
```