PageRank

Geometric Algorithms
Lecture 20

CAS CS 132

Introduction

Recap Problem

$$\begin{bmatrix} 4 & 3 & -1 & 2 & 0 \\ 0 & 2 & -3 & 5 & 1 \\ 0 & 0 & 1 & 3 & -10 \\ 0 & 0 & 0 & -7 & 3 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Determine if the above matrix is diagonalizable.

Answer: Yes

$$\begin{bmatrix} 4 & 3 & -1 & 2 & 0 \\ 0 & 2 & -3 & 5 & 1 \\ 0 & 0 & 1 & 3 & -10 \\ 0 & 0 & 0 & -7 & 3 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Objectives

1. Recall Graphs and Random Walks
2. Connect Random Walks with Markov Chains with Eigenvectors.
3. Discuss PageRank from the perspective of Markov Chains.
4. Learn about the power method as a way to approximate

Keywords

Random Surfer Model

Graphs

Directed vs. Undirected

Weighted vs. Unweighted

Degree

Adjacency Matrices

Spectral/Algebraic Graph Theory

Random Walk

Transition Matrix

Stochastic Matrix

Regular Matrices

Markov Chains

Steady-state vectors

PageRank

Absorbing vs. Reflecting Boundaries

Damping Factor

Power Method

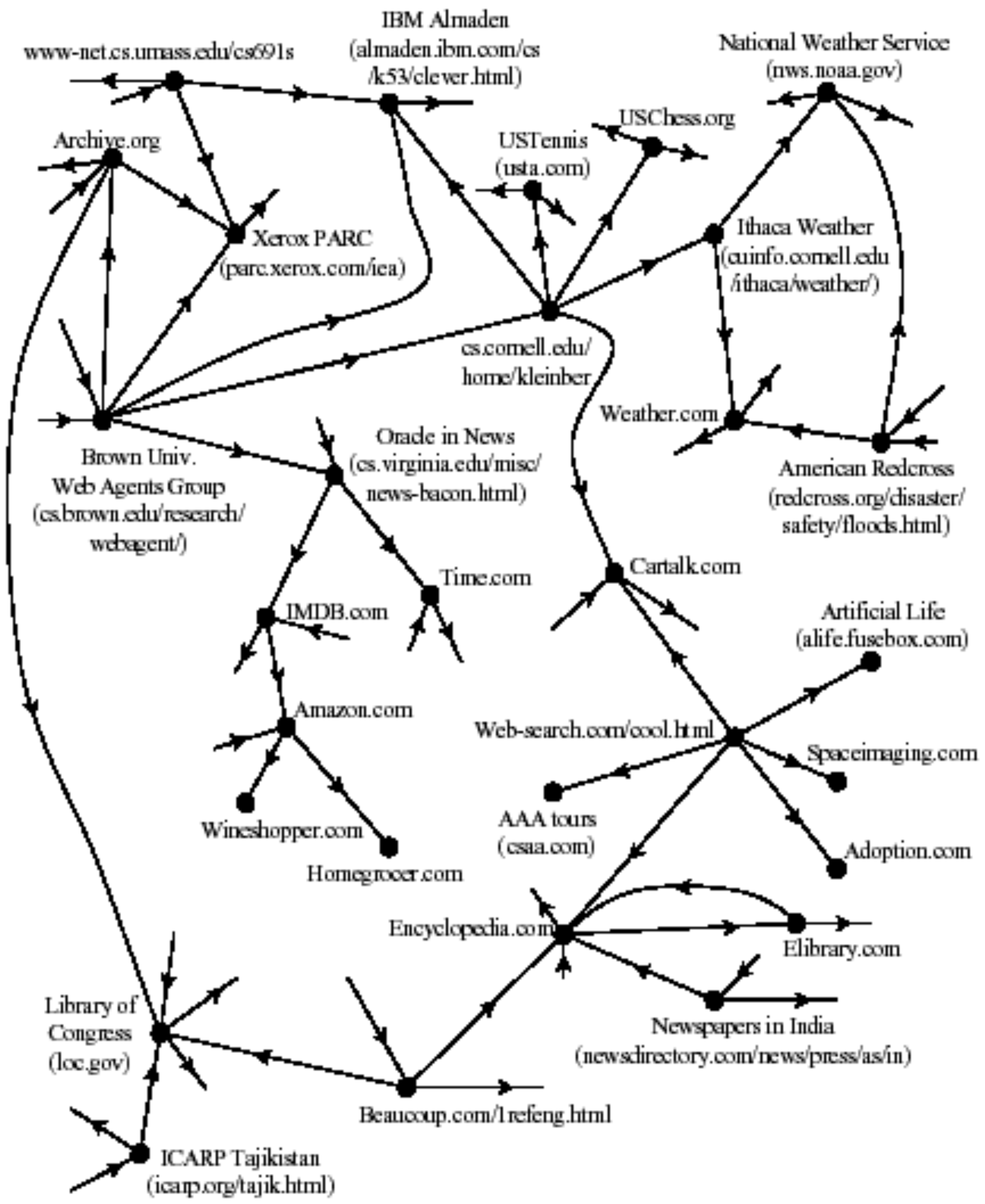
Some "History"

The Web

The World Wide Web is introduced in the 1990s, invented by Tim Berners-Lee.

It has obviously grown in popularity...

At a high level, it is a collection of media (*websites*) connected by directed *hyperlinks*.



https://commons.wikimedia.org/wiki/File:WWW-LetShare.svg
https://inst.eecs.berkeley.edu/~cs61bl/r//cur/graphs/world-wide-web.html?topic=lab24.topic&step=6&course=

The Web

Let's Share What We Know

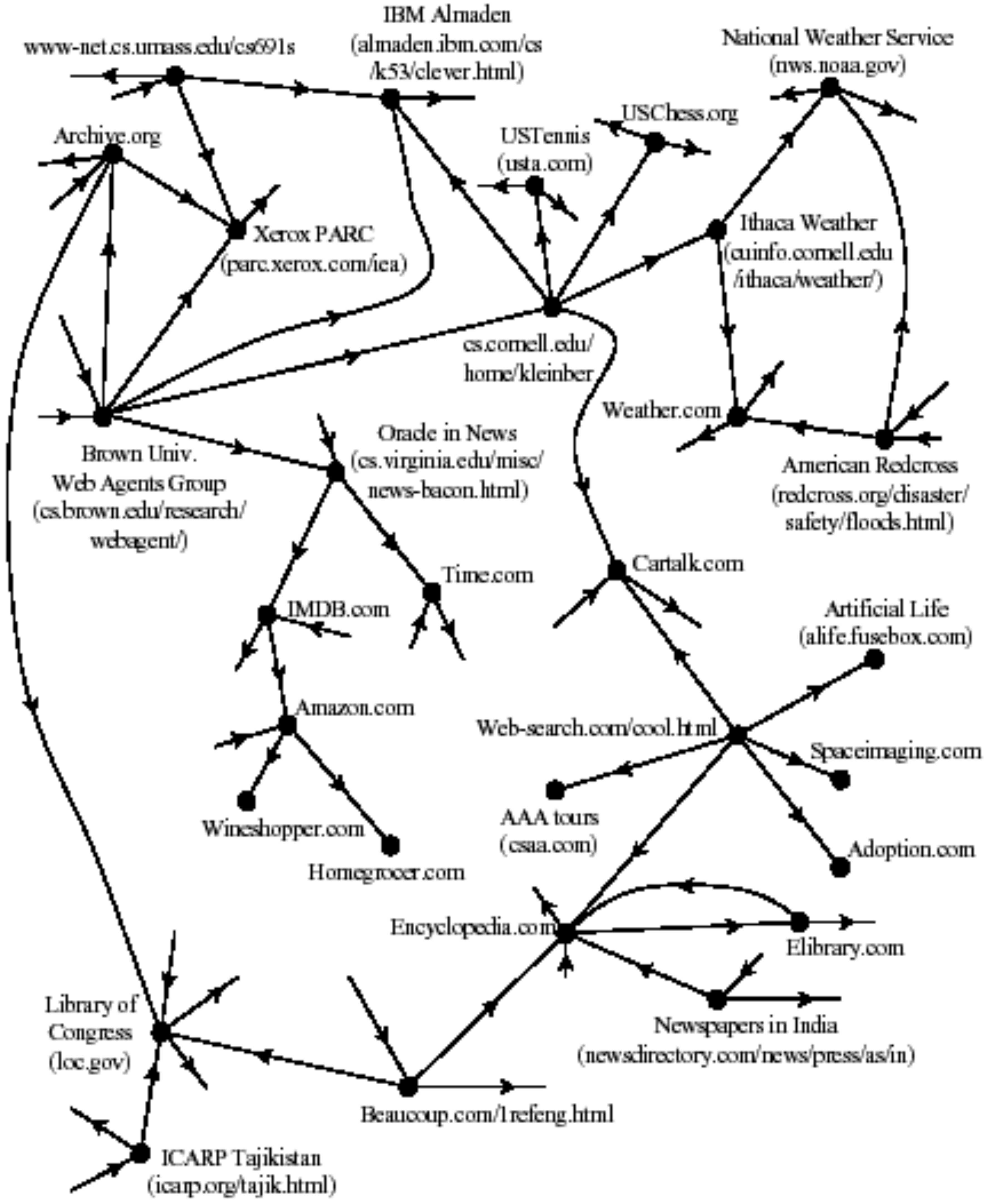
World Wide Web

The World Wide Web is introduced in the 1990s, invented by Tim Berners-Lee.

It has obviously grown in popularity...

At a high level, it is a collection of media (*websites*) connected by directed **Nodes** *hyperlinks*.

Edges



https://commons.wikimedia.org/wiki/File:WWW-LetShare.svg
https://inst.eecs.berkeley.edu/~cs61bl/r//cur/graphs/world-wide-web.html?topic=lab24.topic&step=6&course=

Google

Created by Larry Page and Sergey Brin in 1996 when they were PhD students at Stanford.

Their idea was to build a *search engine*, based on an algorithm they called **PageRank**.

Search Engines

Search Engines

Step 1. Given a search term, find a collection of websites using that term.

Search Engines

Step 1. Given a search term, find a collection of websites using that term.

Step 2. Given a collection of websites based on search term, compute a *ranking* of them by <u>importance</u> (the most important websites should be presented first).

Search Engines

Step 1. Given a search term, find a collection of websites using that term.

Step 2. Given a collection of websites based on search term, compute a *ranking* of them by <u>importance</u> (the most important websites should be presented first).

How do we know which websites are important?

Ranking Websites

https://www.cs.cornell.edu/~kt/post/site-graph/

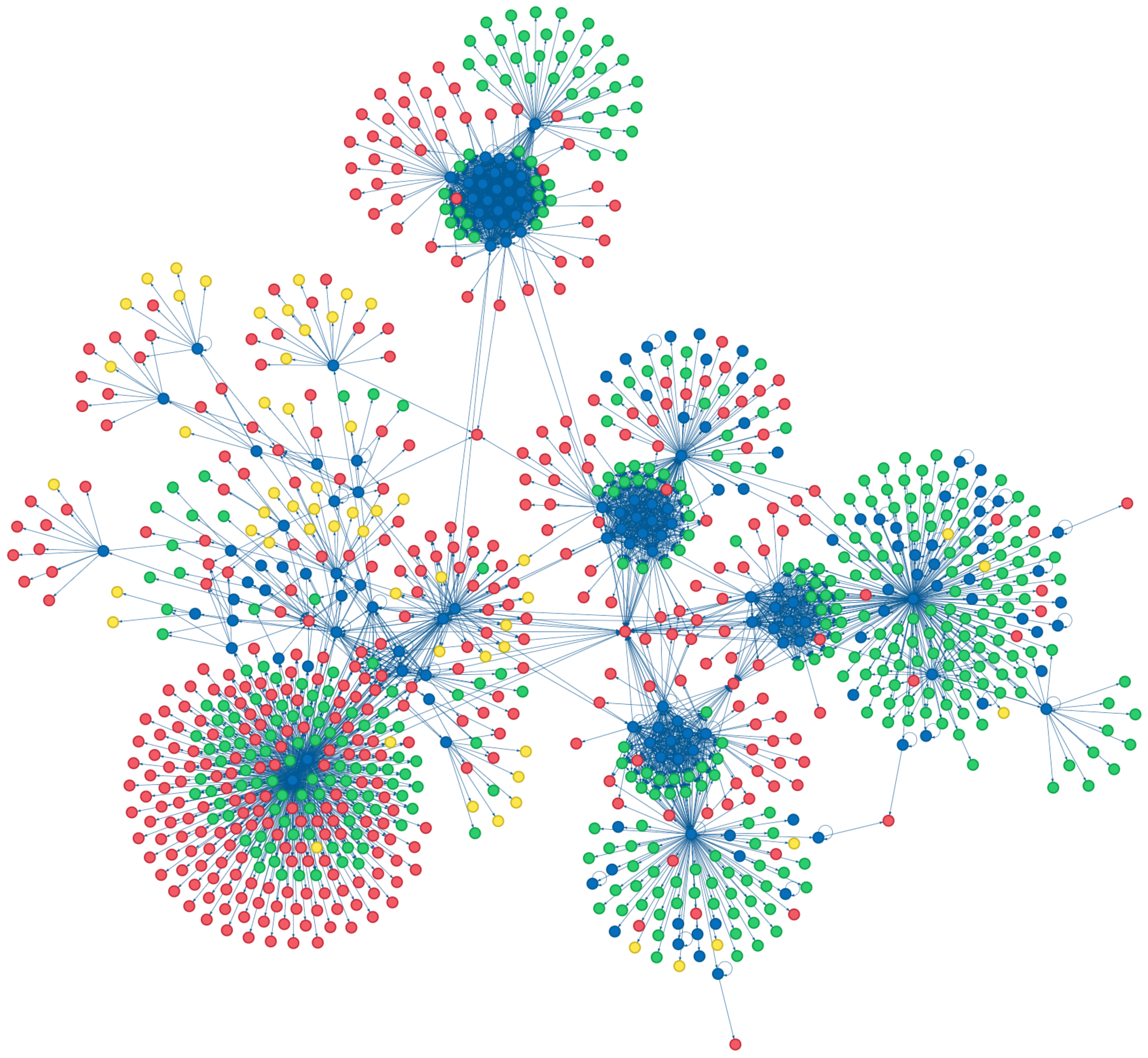
Ranking Websites

Idea 1. (*Term frequency*) If your search term is used many times on a page, it is likely an important page for that term.

Ranking Websites

Idea 1. (*Term frequency*) If your search term is used many times on a page, it is likely an important page for that term.

Idea 2. *(Linking Structure)* If is a site is **linked a bunch of times**, it is an important page



https://www.cs.cornell.edu/~kt/post/site-graph/

The Random Surfer Model

2.1.2. Intuitive justification

PageRank can be thought of as a model of user behavior. We assume there is a "random surfer" who is given a Web page at random and keeps clicking on links, never hitting "back" but eventually gets bored and starts on another random page. The probability that the random surfer visits a page is its PageRank.

The anatomy of a large-scale hypertextual Web search engine (1998)

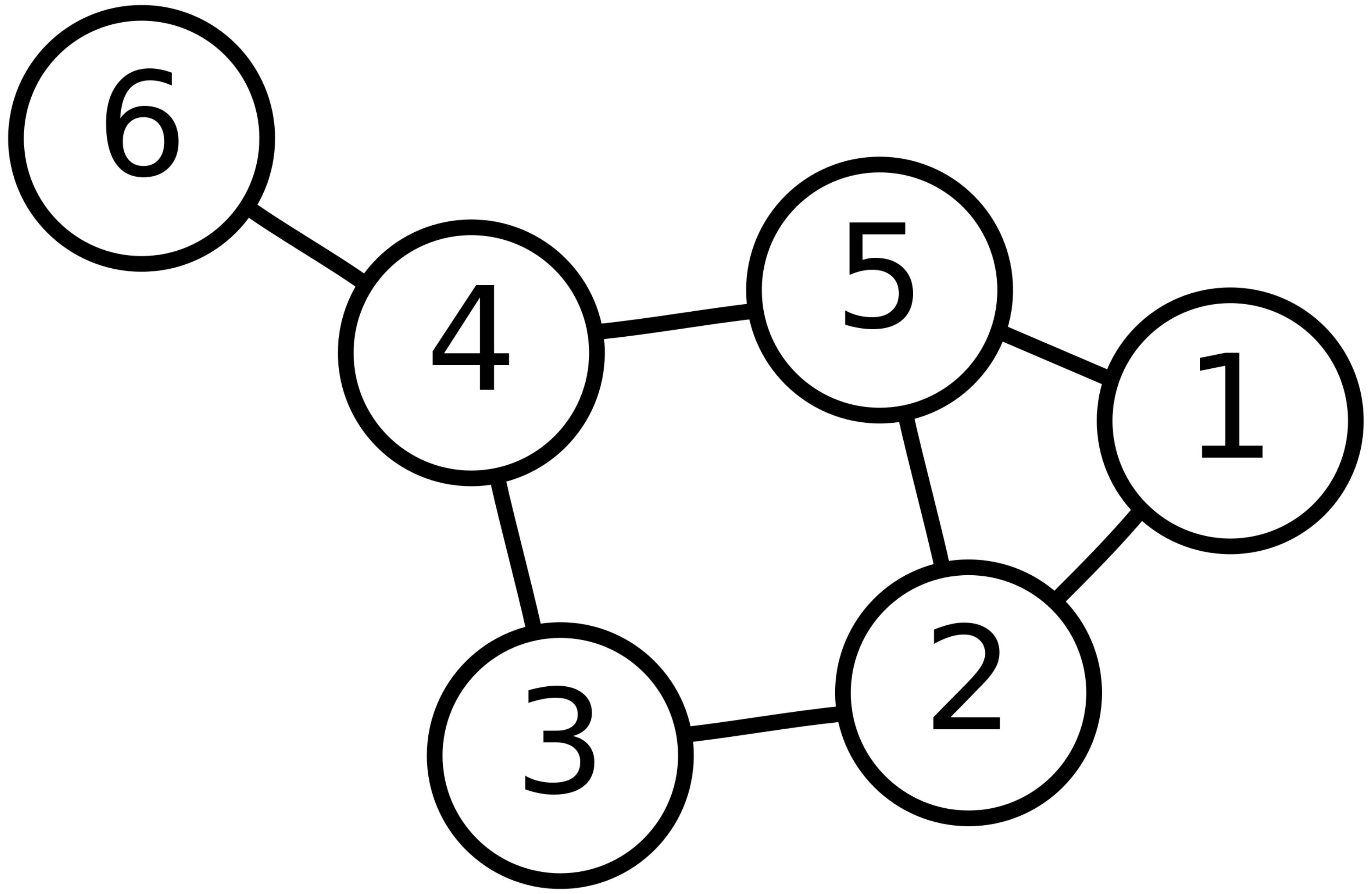
This is really just a **random walk** on a **directed graph**

(which is really just a **Markov Chain**)

Graphs

Recall: Graphs

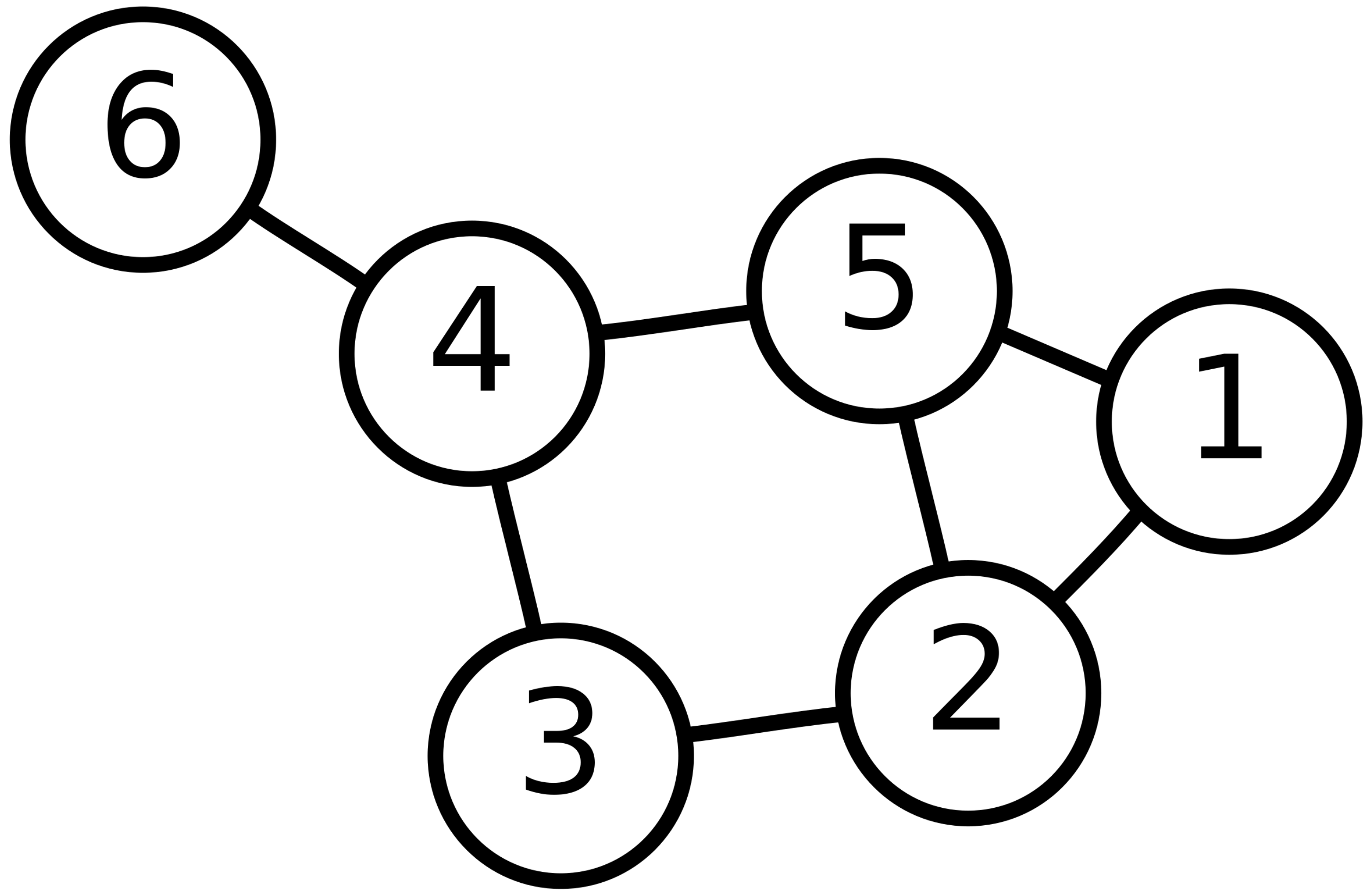
Definition (Informal). A **graph** is a collection of nodes with edges between them.



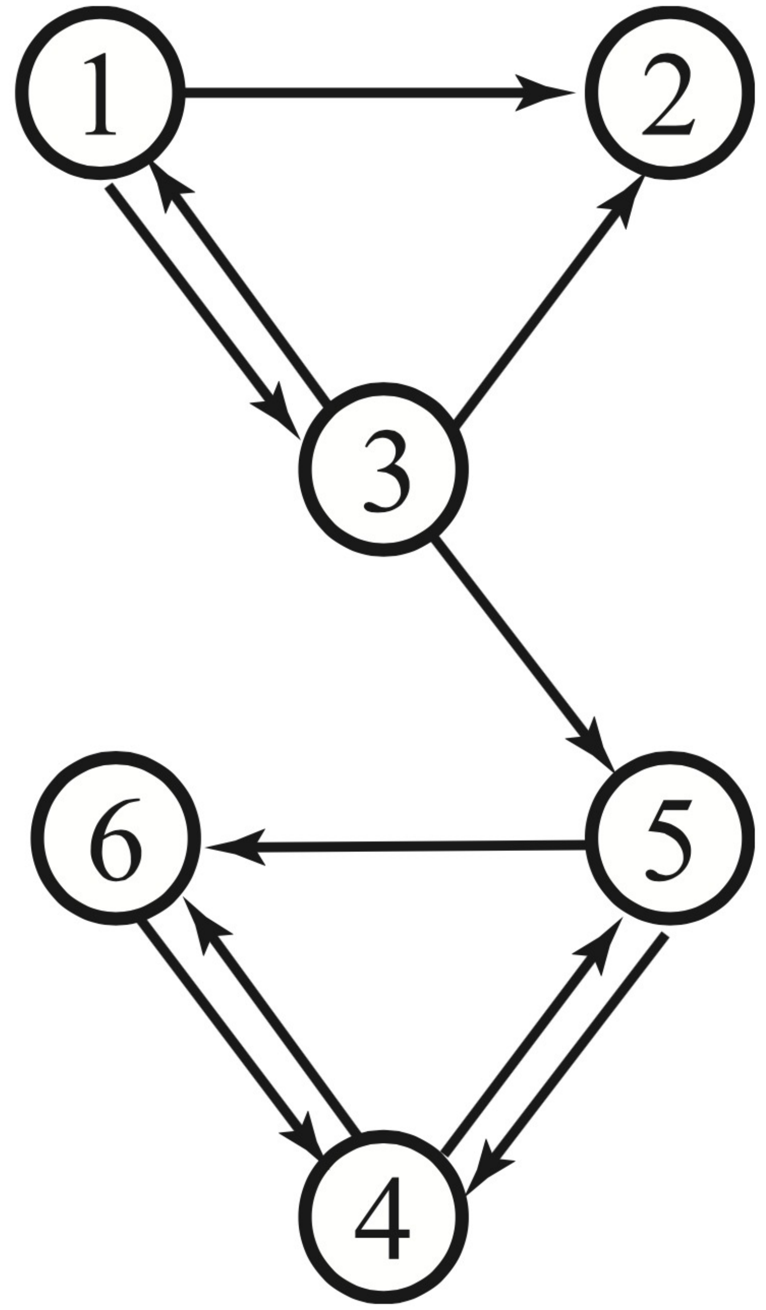
https://commons.wikimedia.org/wiki/File:6n-graf.svg

Directed vs. Undirected Graphs

A graph is **directed** if its edges have a direction.



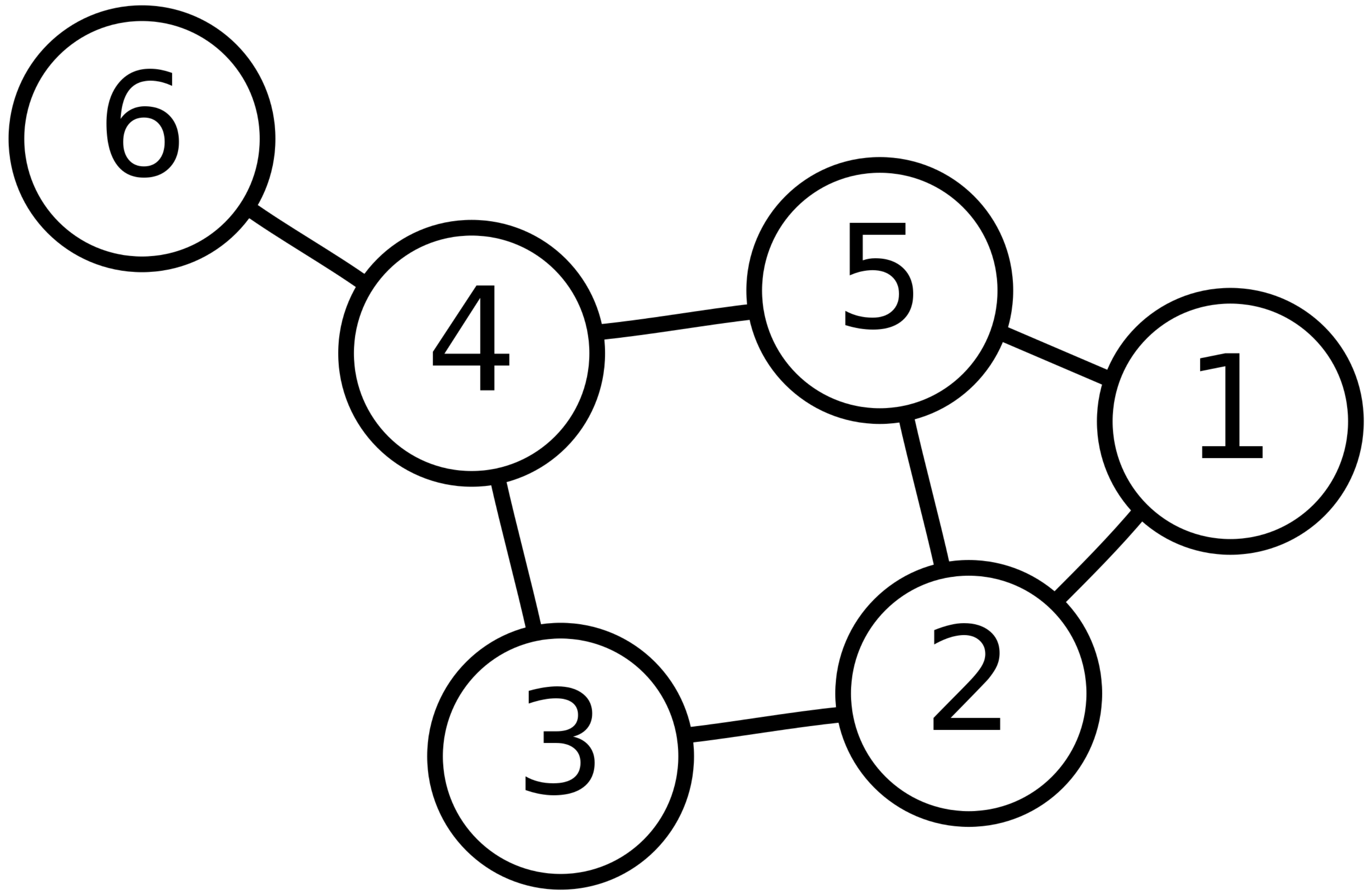
undirected



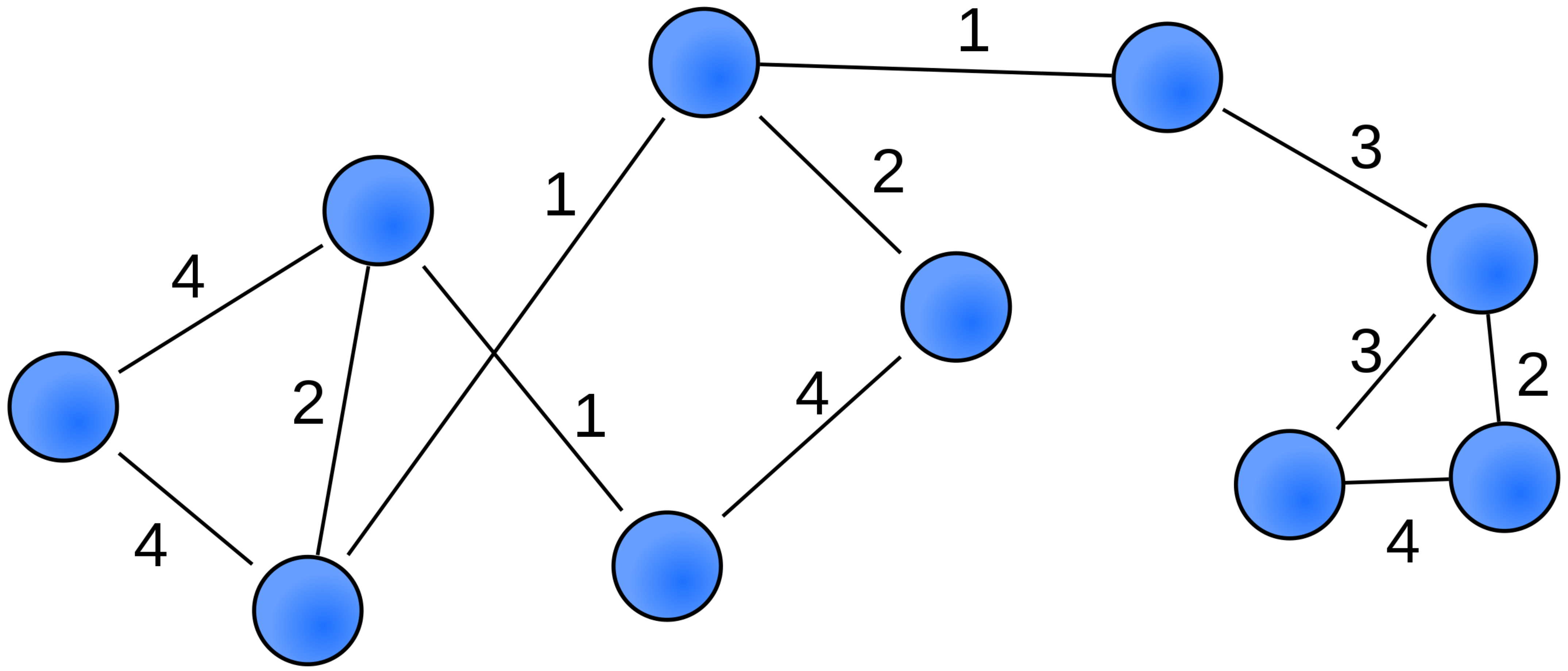
directed

Weighted vs Unweighted graphs

A graph is **weighted** if its edges have associated values.



unweighted

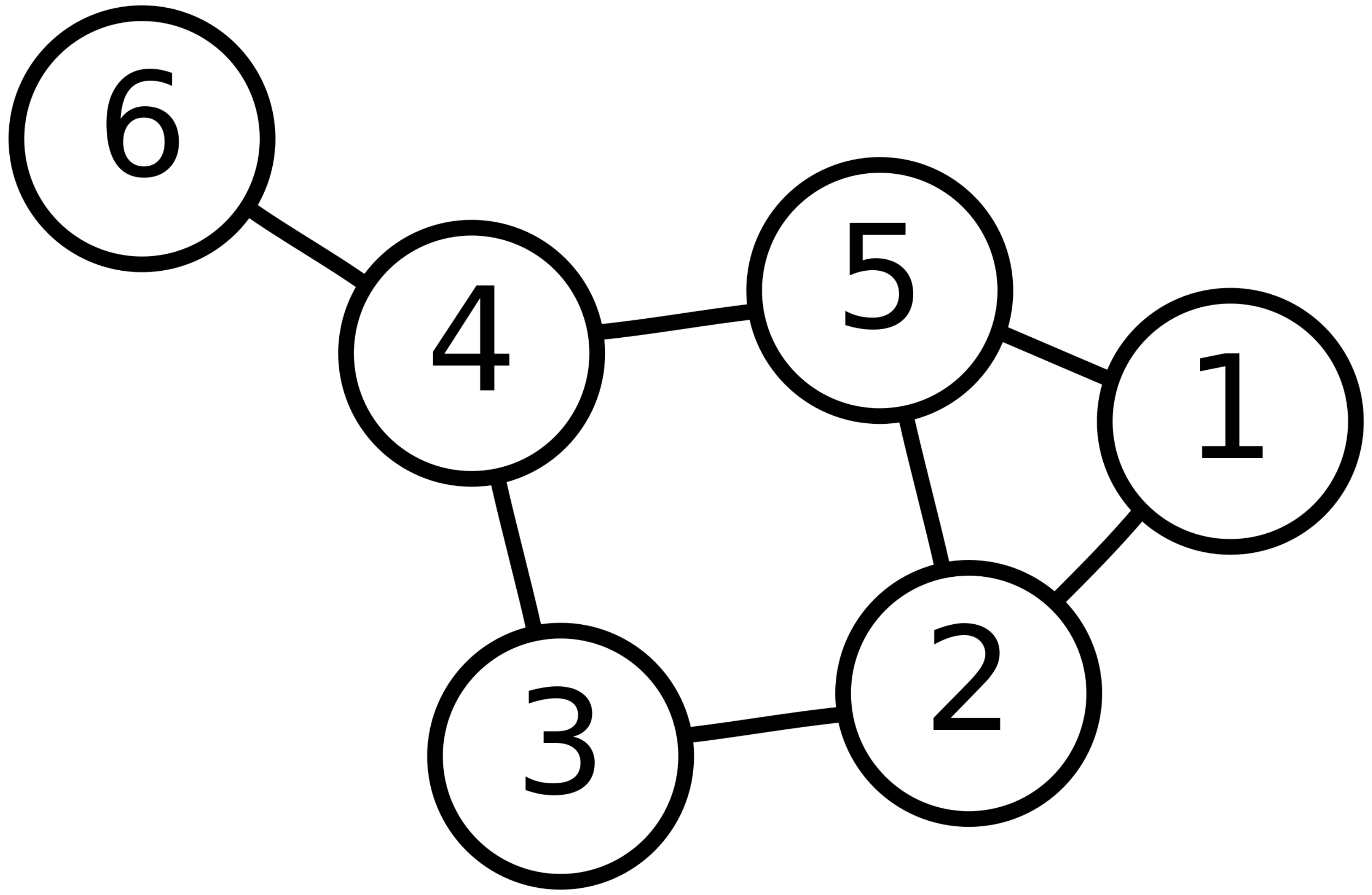


weighted

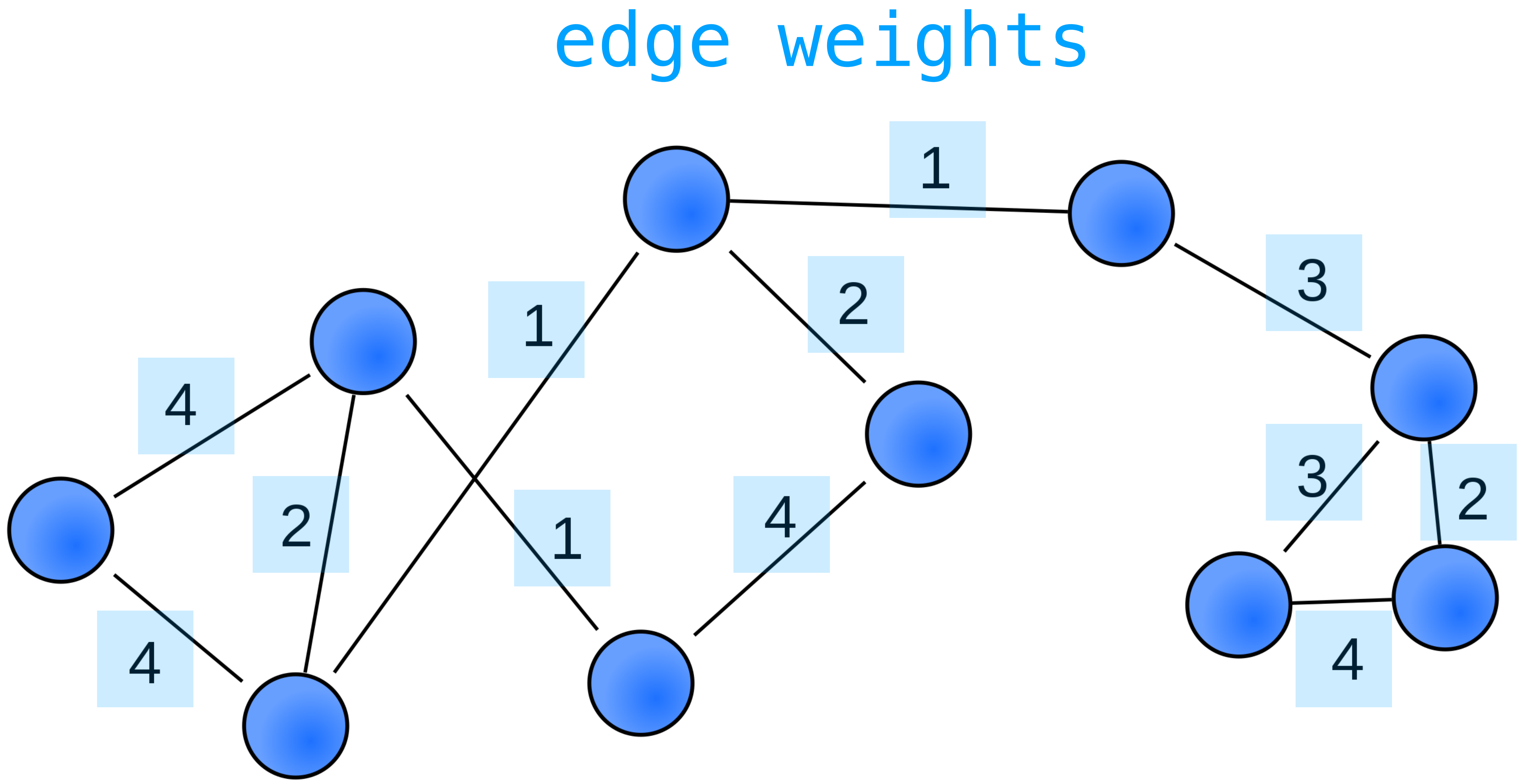
https://commons.wikimedia.org/wiki/File:Weighted_network.svg

Weighted vs Unweighted graphs

A graph is **weighted** if its edges have associated values.



unweighted



weighted

https://commons.wikimedia.org/wiki/File:Weighted_network.svg

Four Kinds of Graphs

	directed	undirected
weighted	nodes are traffic lights edges are streets weights are number of lanes	nodes are musicians edges are collaborations weights are number of collaborations
unweighted	nodes are instagram users edges are follows	nodes are bodies of land edges are pedestrian bridges

Four Kinds of Graphs

	directed	undirected
weighted	nodes are traffic lights edges are streets weights are number of lanes	nodes are musicians edges are collaborations weights are number of collaborations
unweighted	nodes are instagram users edges are follows The Web	nodes are bodies of land edges are pedestrian bridges

Four Kinds of Graphs

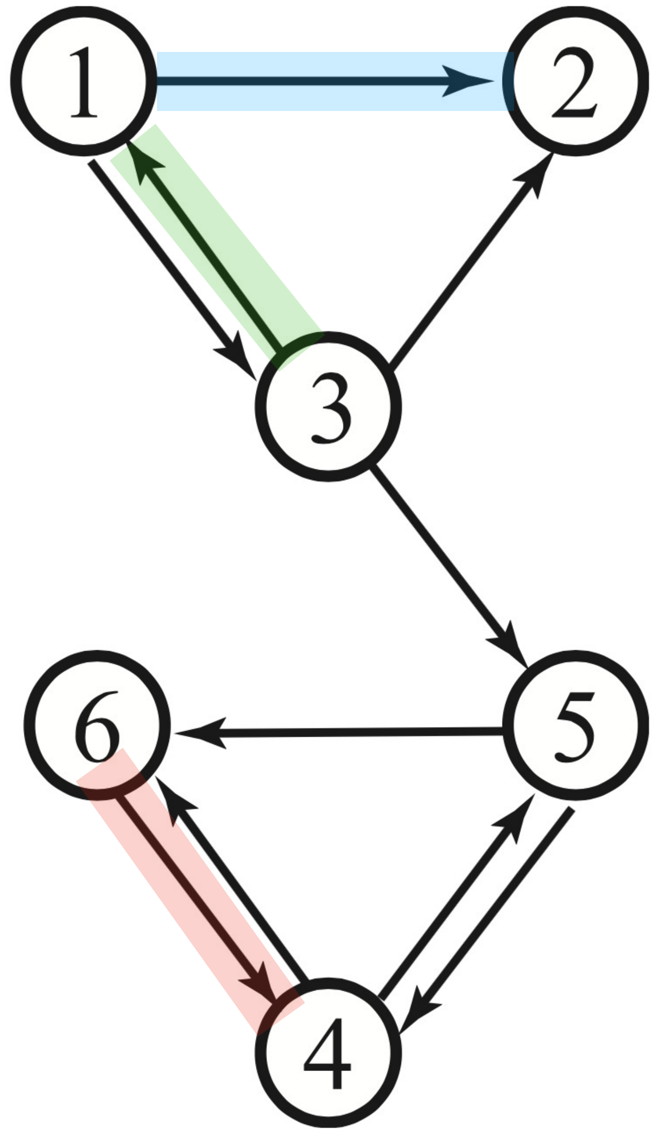
	directed	undirected
weighted	nodes are traffic lights edges are streets weights are number of lanes Markov Chains	nodes are musicians edges are collaborations weights are number of collaborations
unweighted	nodes are instagram users edges are follows The Web	nodes are bodies of land edges are pedestrian bridges

Recall: Adjacency Matrices

$$
\begin{array}{ccc}
 & \begin{array}{cccccc} & & A_{13} & & & \end{array} & \\
\begin{array}{c} \\ \\ A_{21} \\ \\ \\ \\ \end{array} & \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix} & \begin{array}{c} \\ \\ \\ A_{46} \\ \\ \\ \end{array}
\end{array}
$$

Let G be an **directed** graph with its nodes labeled by numbers 1 through n.

We can create the **adjacency matrix** A for G as follows.



$$A_{ij} = \begin{cases} 1 & \text{there is an edge from j and i} \\ 0 & \text{otherwise} \end{cases}$$

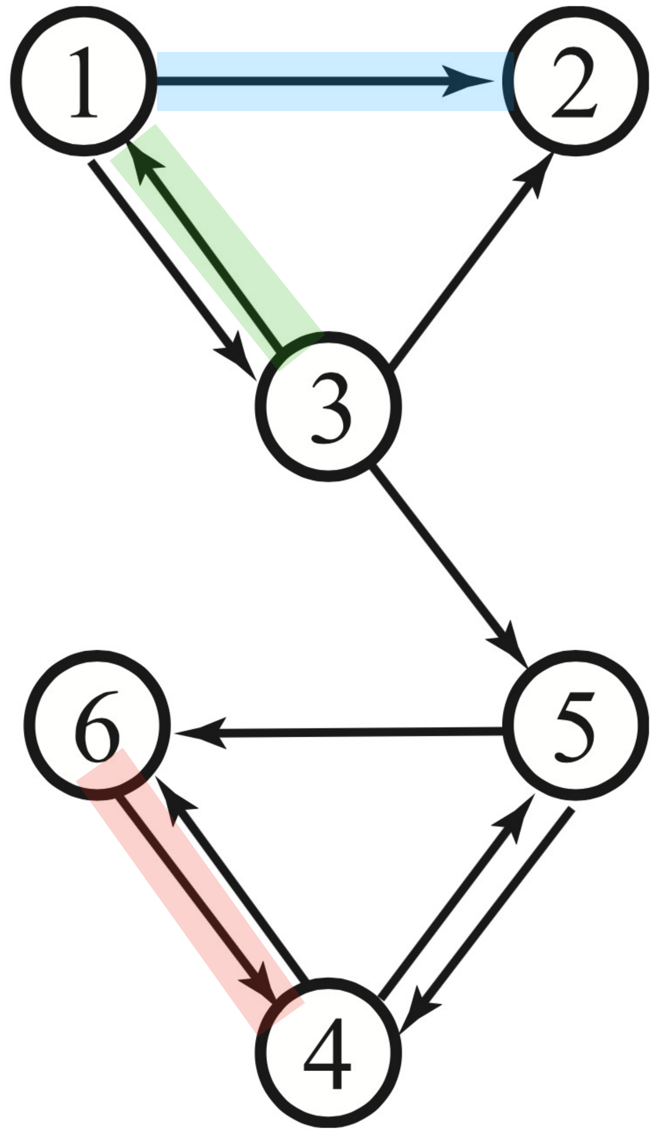
Recall: Adjacency Matrices

$$\begin{array}{ccc} & A_{13} & \\ A_{21} & \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix} & A_{46} \end{array}$$

represents edges out of 1

Let G be an **directed** graph with its nodes labeled by numbers 1 through n.

We can create the **adjacency matrix** A for G as follows.

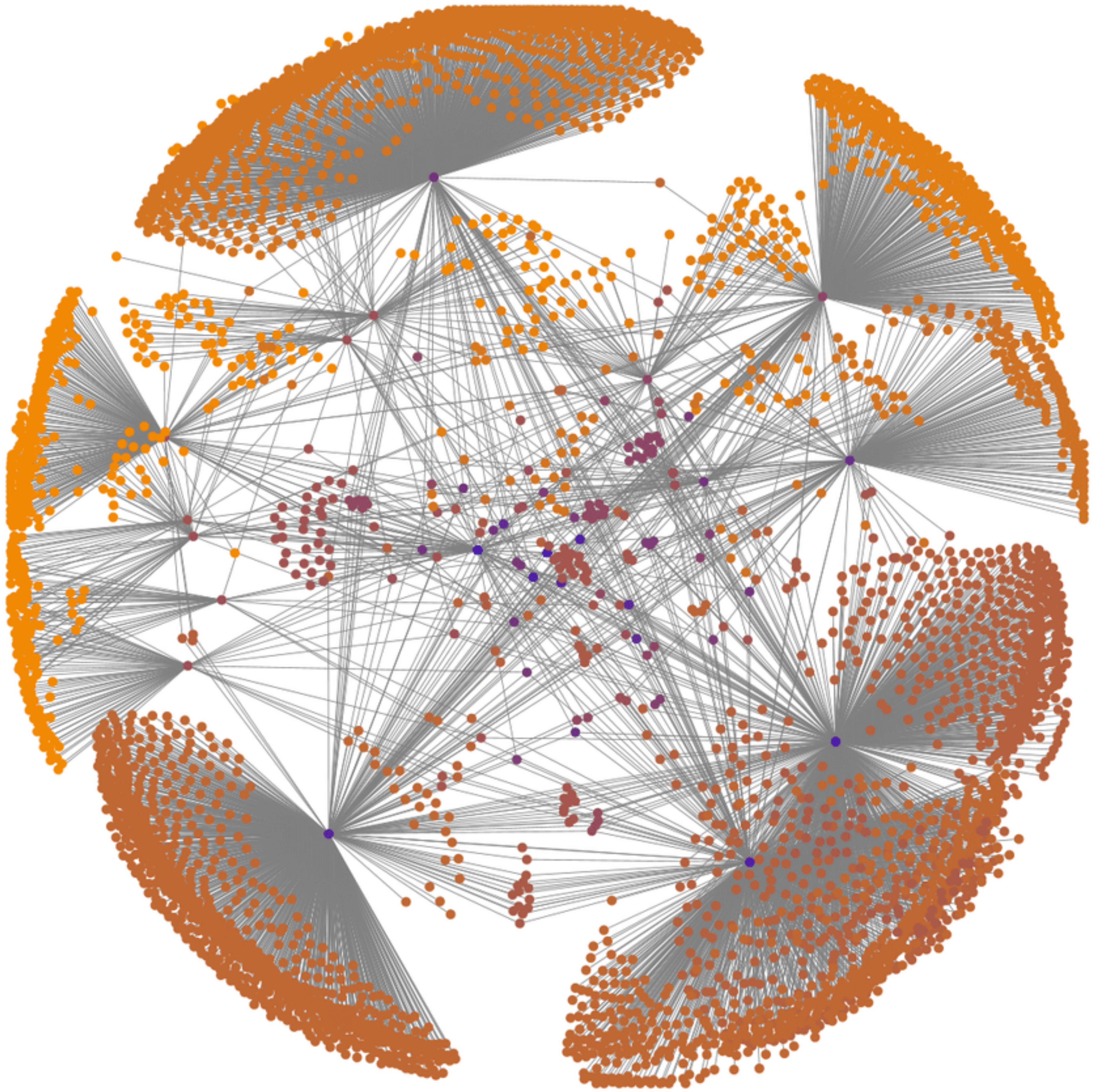


$$A_{ij} = \begin{cases} 1 & \text{there is an edge from j and i} \\ 0 & \text{otherwise} \end{cases}$$

Spectral/Algebraic Graph Theory

Once we have an adjacency matrix, we can do linear algebra on graphs.

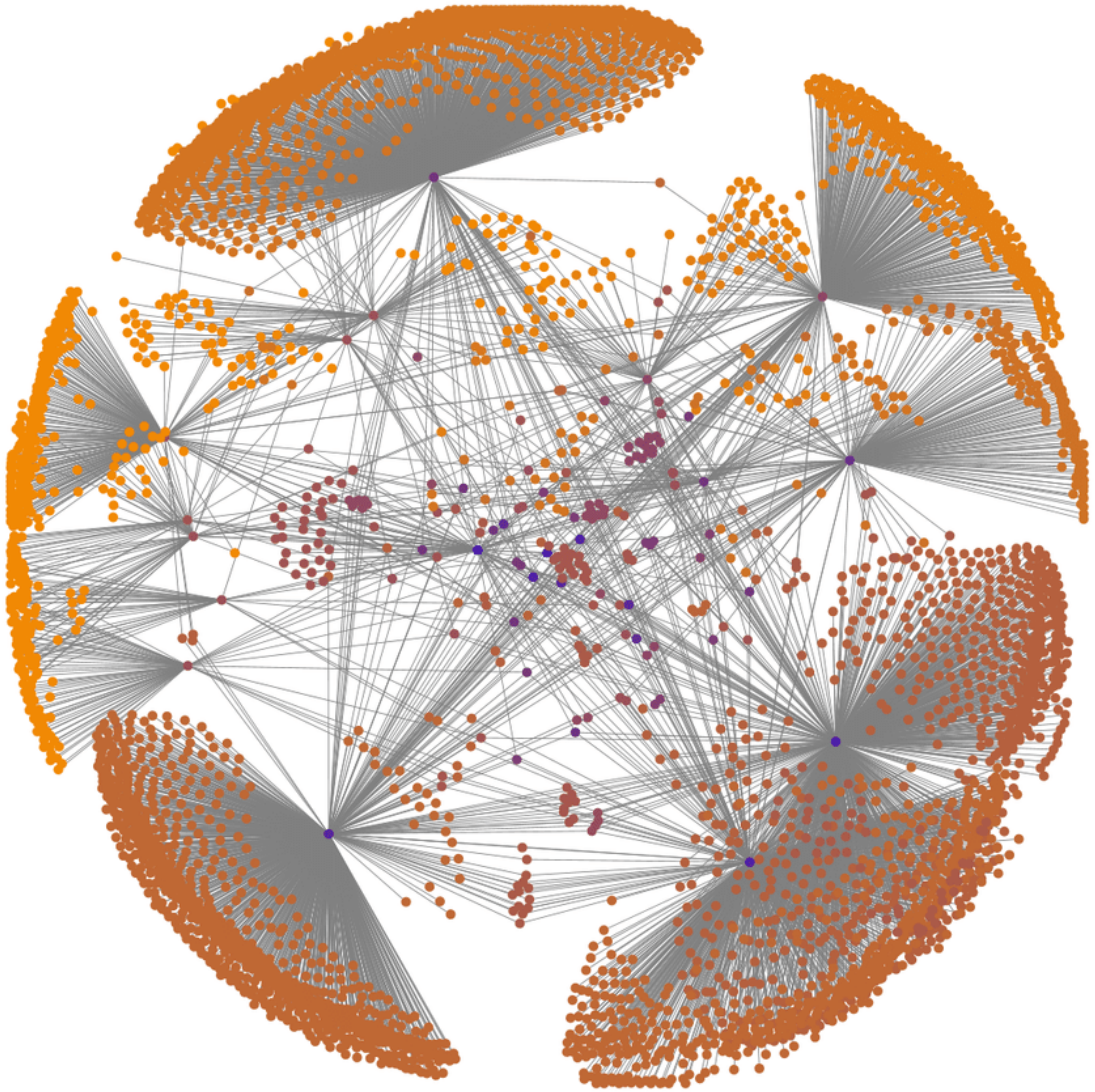
Connecting Back to PageRank



https://www.researchgate.net/figure/This-visualization-represents-the-network-of-web-pages-connected-to-vosonanueduau_fig4_262172827

Connecting Back to PageRank

The Web is a massive **directed graph.**

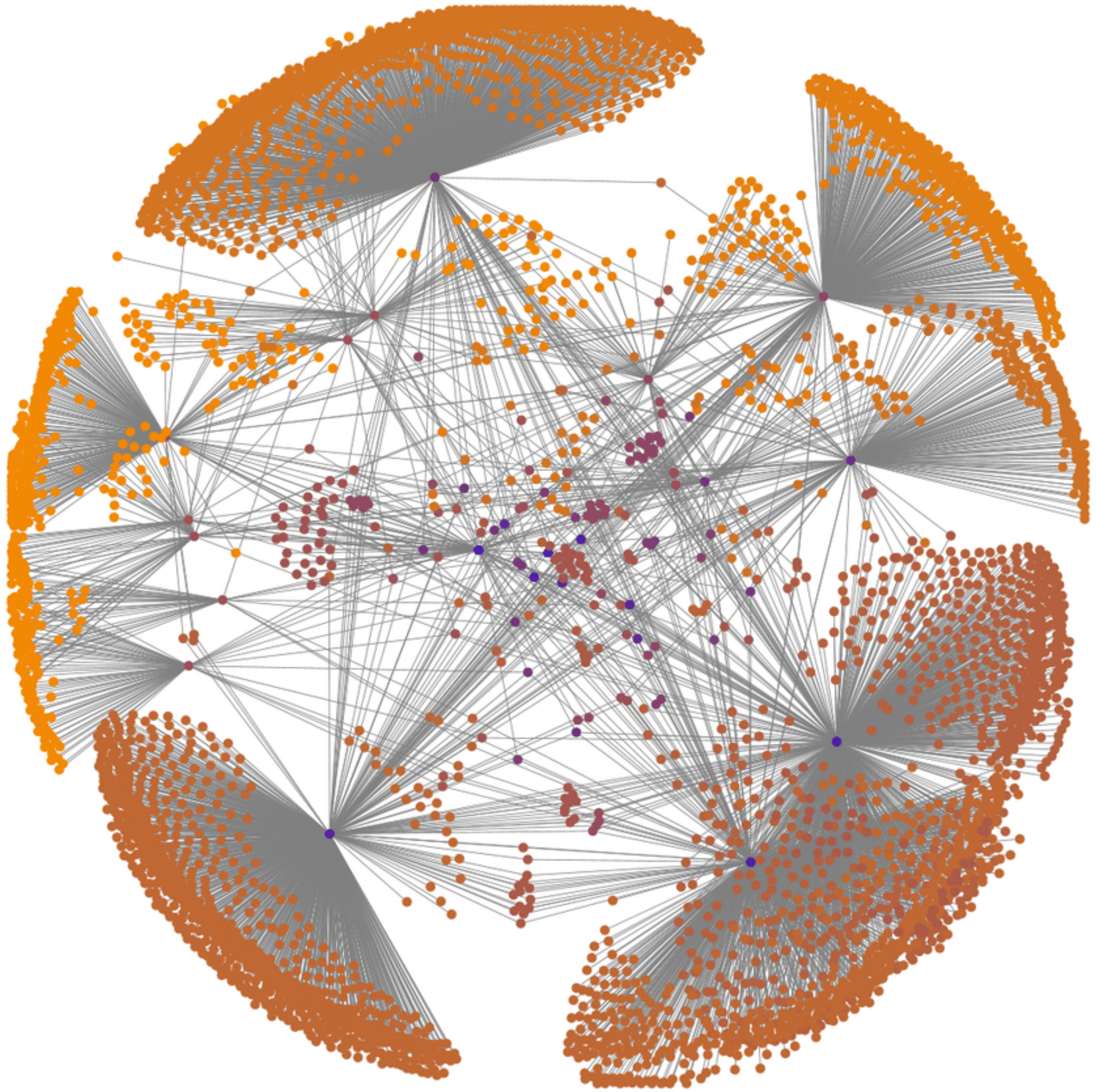


https://www.researchgate.net/figure/This-visualization-represents-the-network-of-web-pages-connected-to-vosonanueduau_fig4_262172827

Connecting Back to PageRank

The Web is a massive **directed graph.**

We will represent the surfer as a <u>random</u> process which explores this graph.



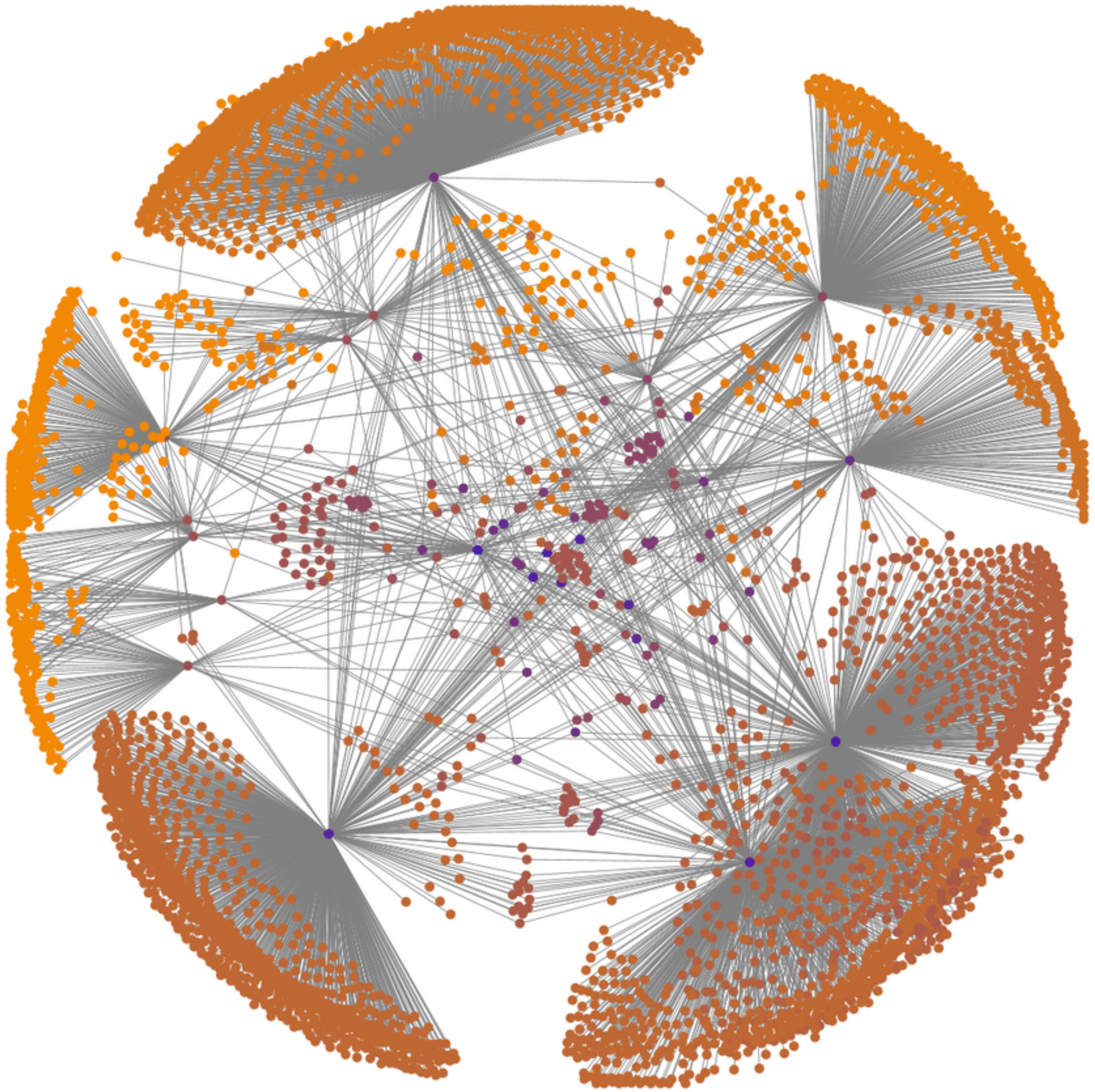
https://www.researchgate.net/figure/This-visualization-represents-the-network-of-web-pages-connected-to-vosonanueduau_fig4_262172827

Connecting Back to PageRank

The Web is a massive **directed graph.**

We will represent the surfer as a random process which explores this graph.

Which connects us back to Markov chains...



https://www.researchgate.net/figure/This-visualization-represents-the-network-of-web-pages-connected-to-vosonanueduau_fig4_262172827

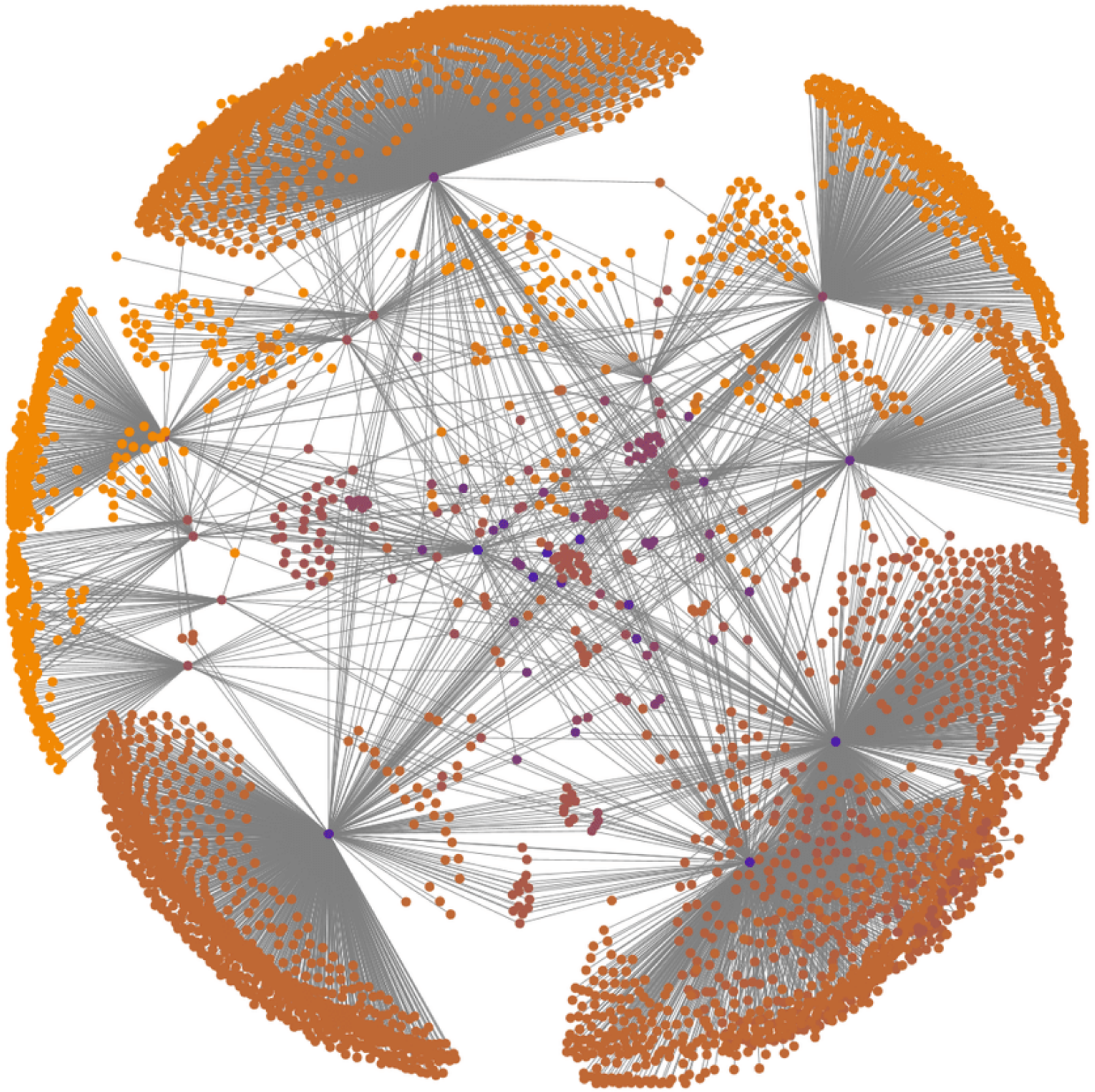
Connecting Back to PageRank

The Web is a massive **directed graph.**

We will represent the surfer as a <u>random</u> process which explores this graph.

Which connects us back to Markov chains...

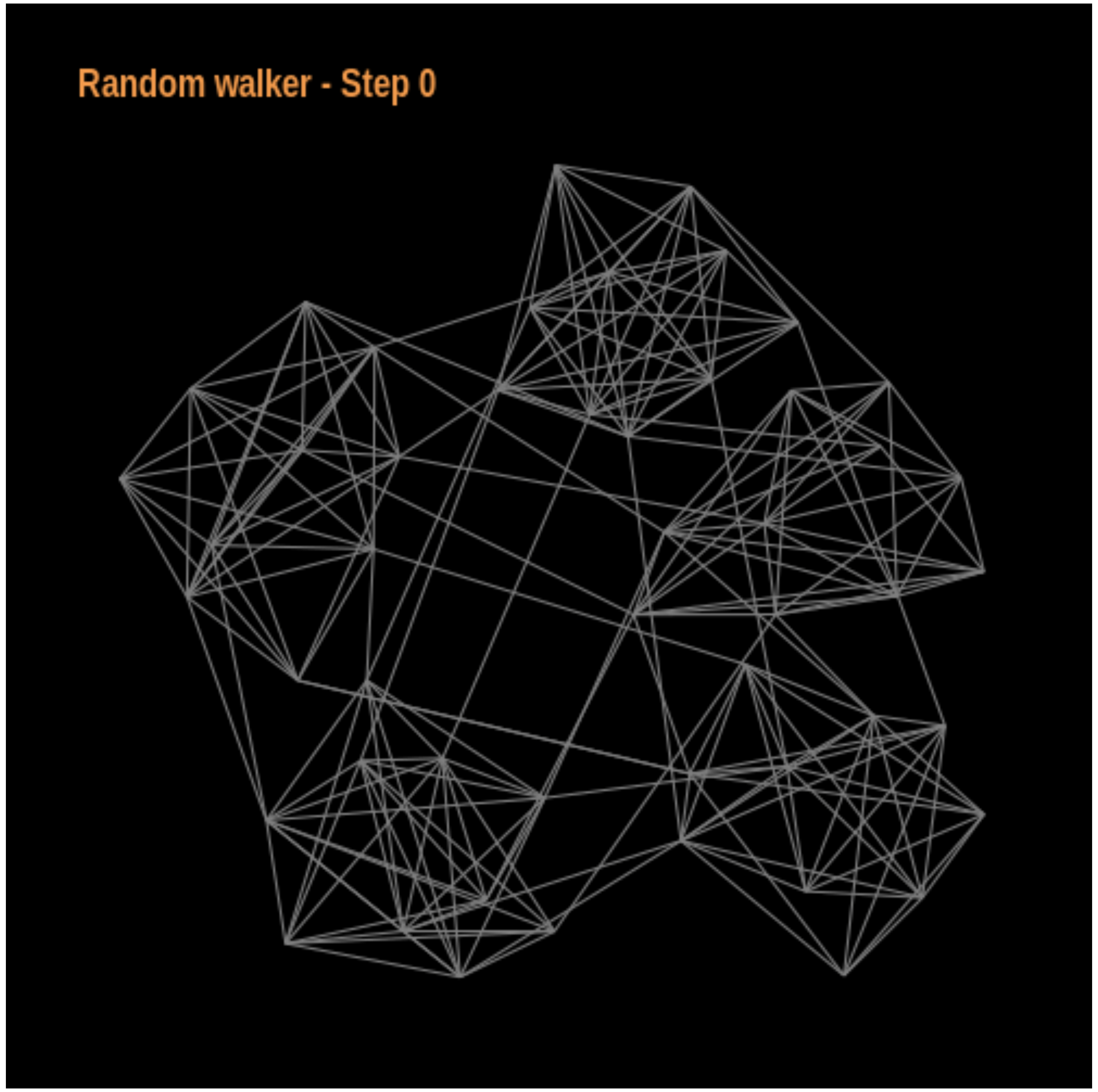
which connects us back to eigenvectors...

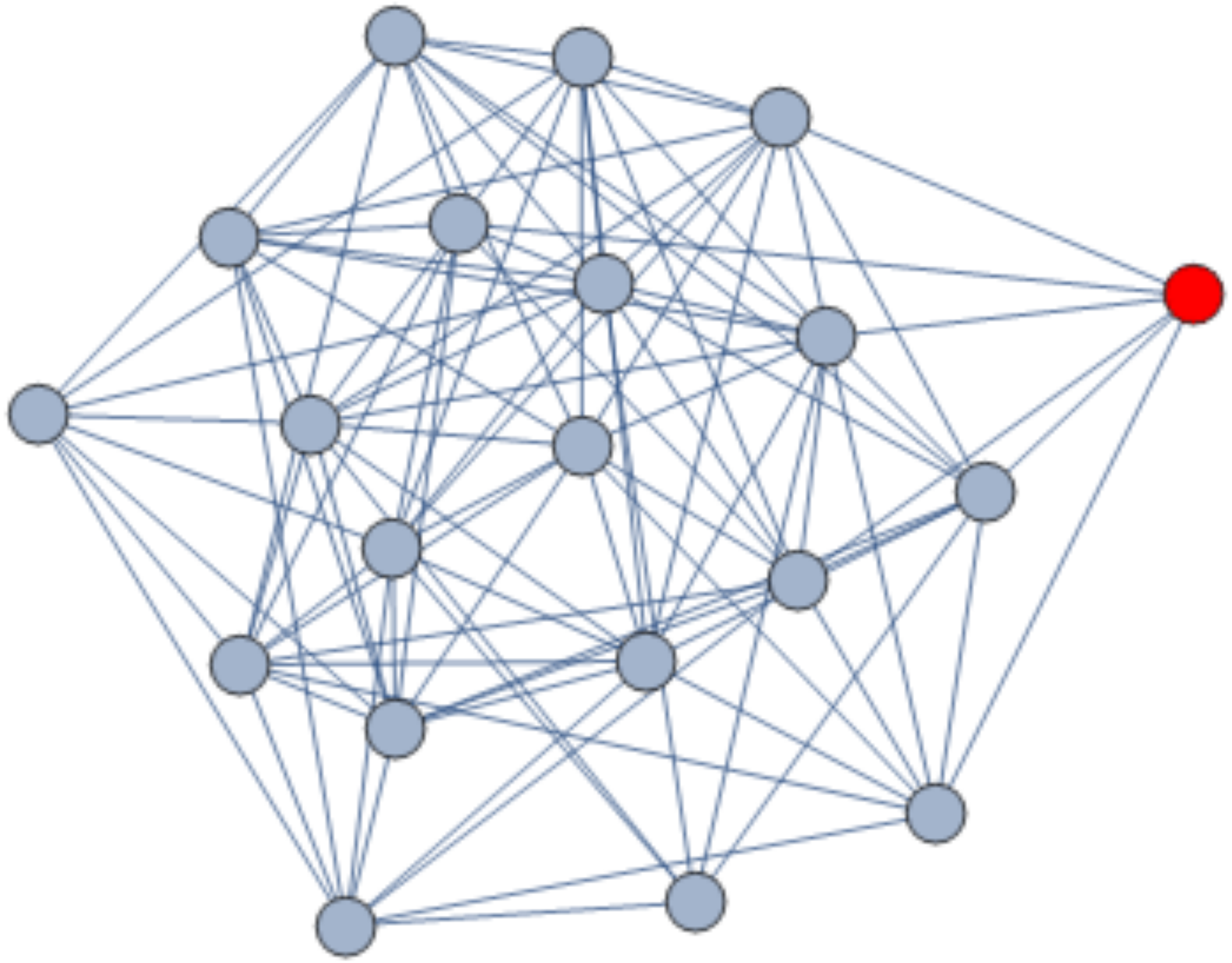


https://www.researchgate.net/figure/This-visualization-represents-the-network-of-web-pages-connected-to-vosonanueduau_fig4_262172827

Random Walks

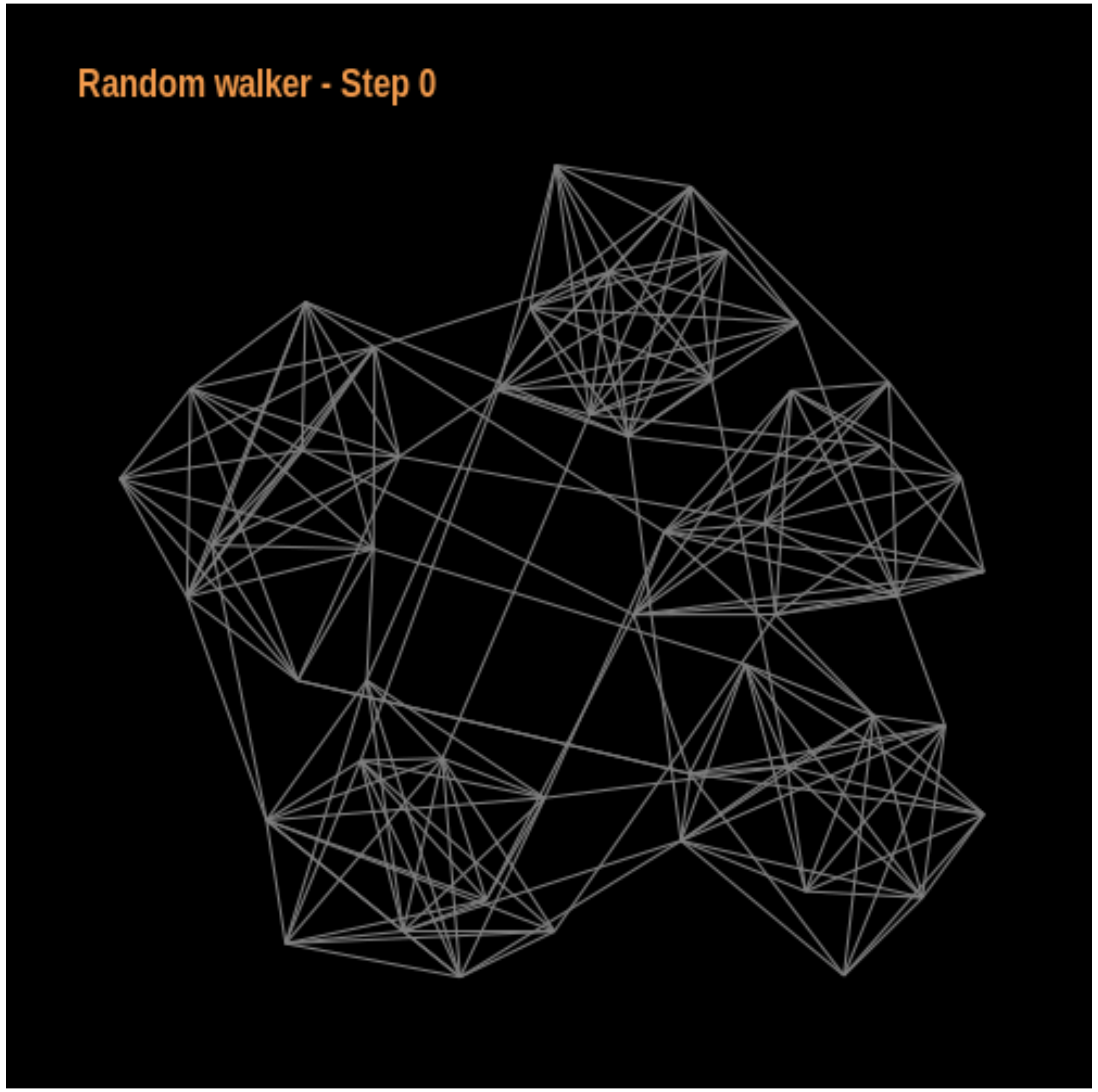
Visualization (In Undirected Case)

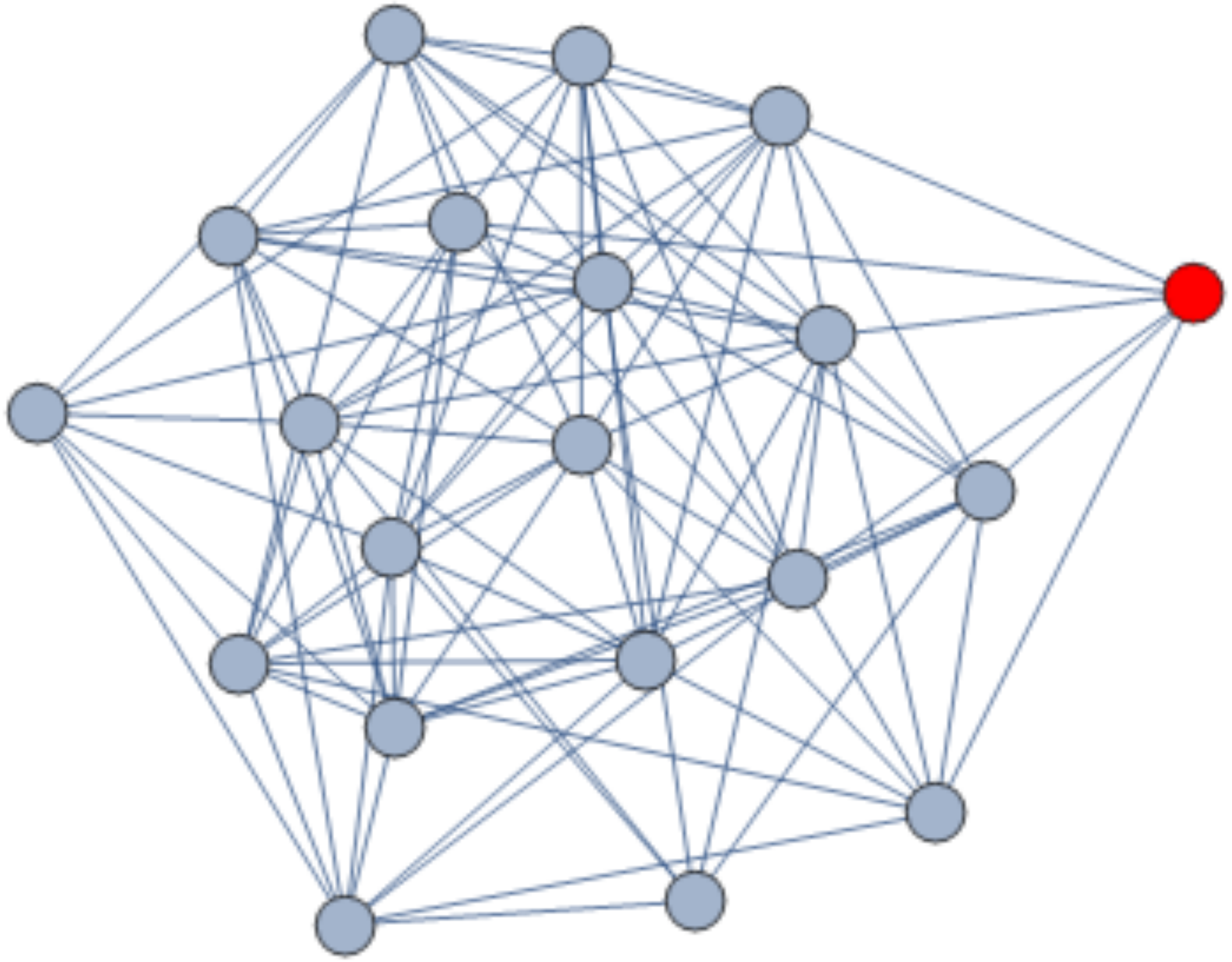




https://mathematica.stackexchange.com/questions/156626/generate-random-walk-on-a-graph
https://gist.github.com/clairemcwhite/7fb348acca2c84c464d751ba38ce72e1

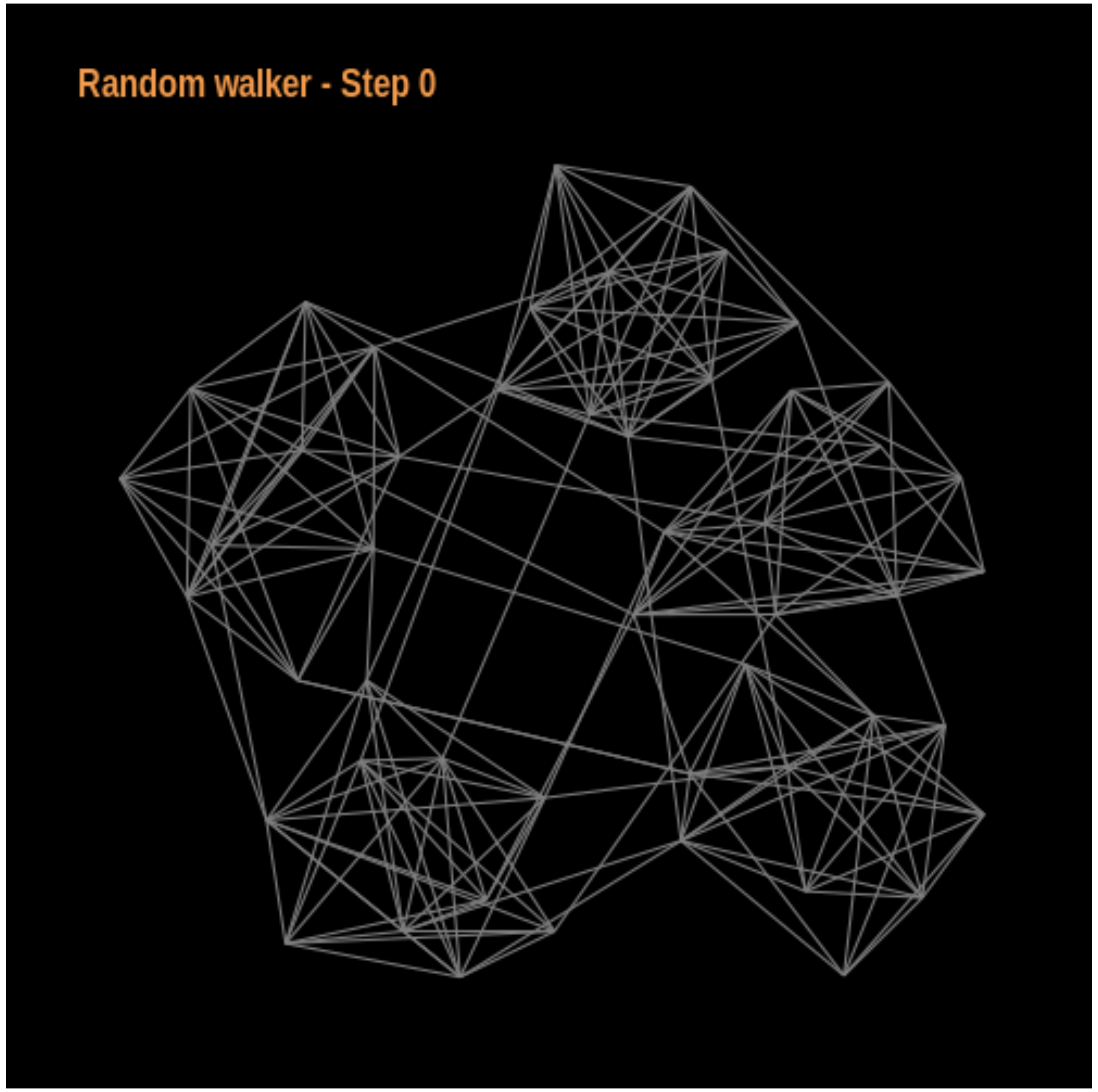
Visualization (In Undirected Case)

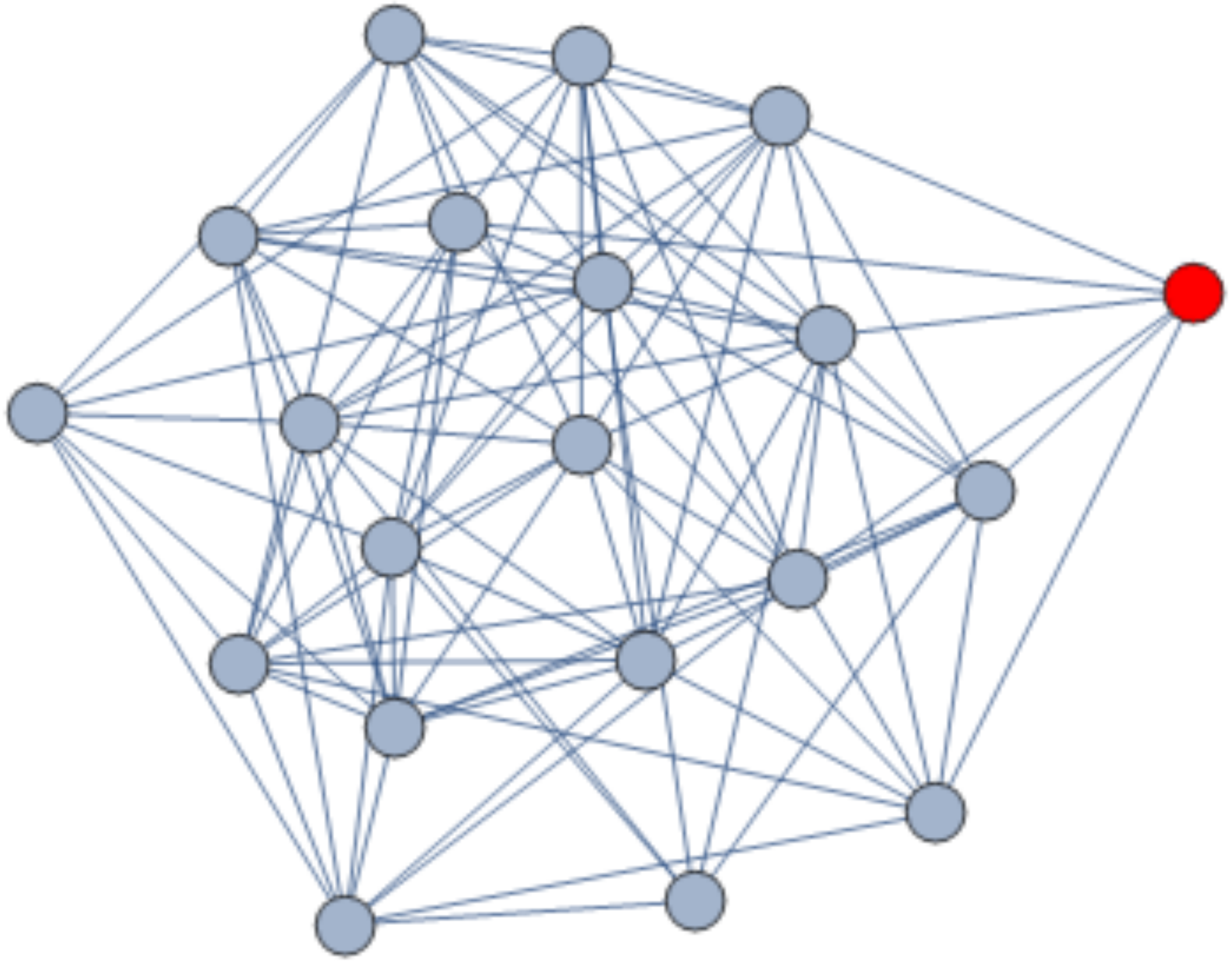




https://mathematica.stackexchange.com/questions/156626/generate-random-walk-on-a-graph
https://gist.github.com/clairemcwhite/7fb348acca2c84c464d751ba38ce72e1

Visualization (In Undirected Case)





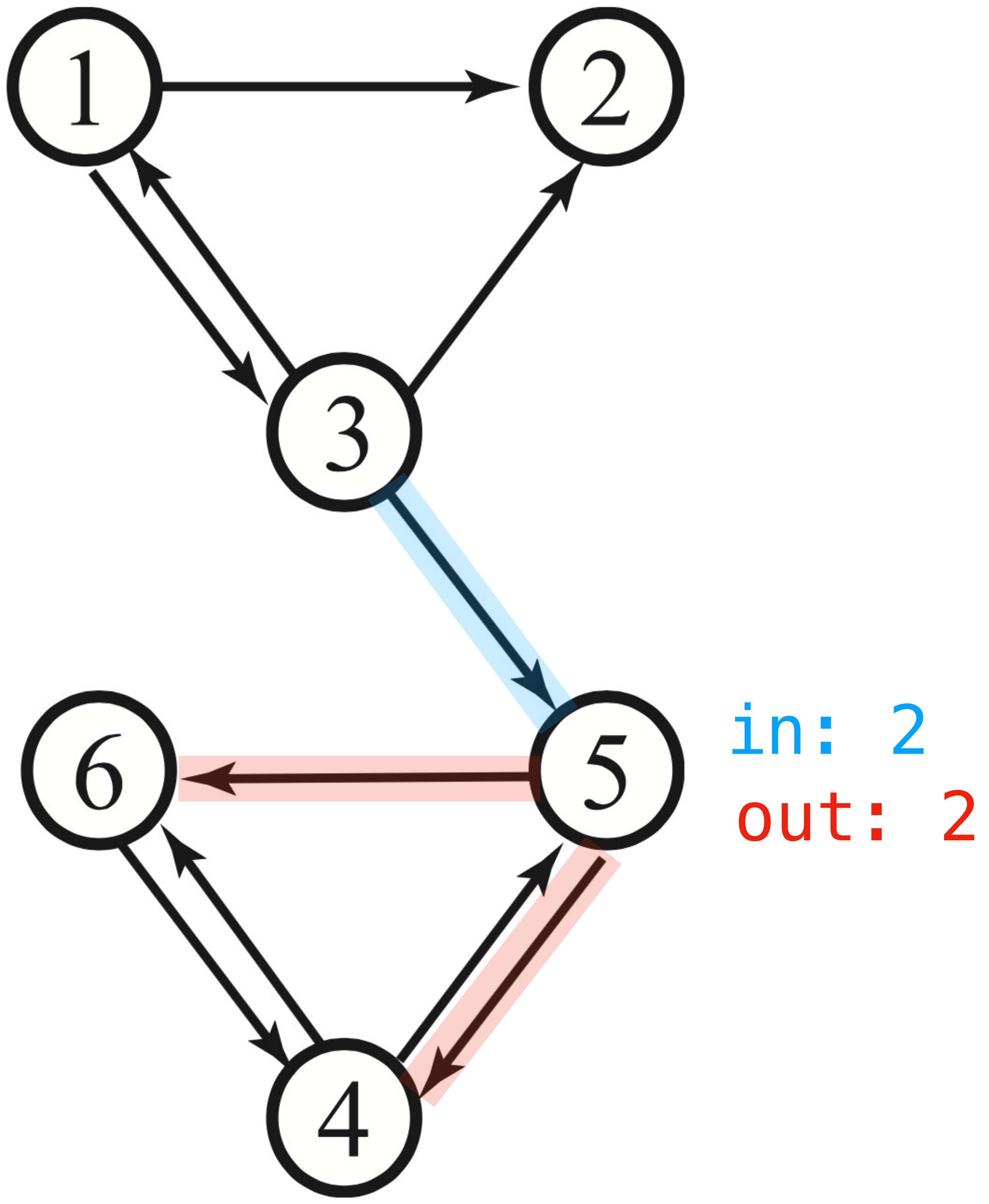
https://mathematica.stackexchange.com/questions/156626/generate-random-walk-on-a-graph
https://gist.github.com/clairemcwhite/7fb348acca2c84c464d751ba38ce72e1

Terminology: Degree

Let G be an unweighted directed graph and let v be one of its nodes.

The **in-degree** of v is the number of edges whose right endpoint is v (that *go into* v)

The **out-degree** of v is the number of edges whose left endpoint is v (that *exit out of* v).



The Procedure

The Procedure

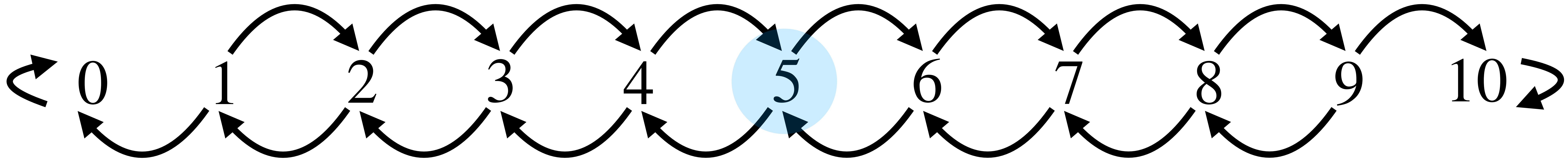
Definition. A **random walk** on an unweighted directed graph G with nodes $\{1,...,n\}$ starting at v is the following process:

The Procedure

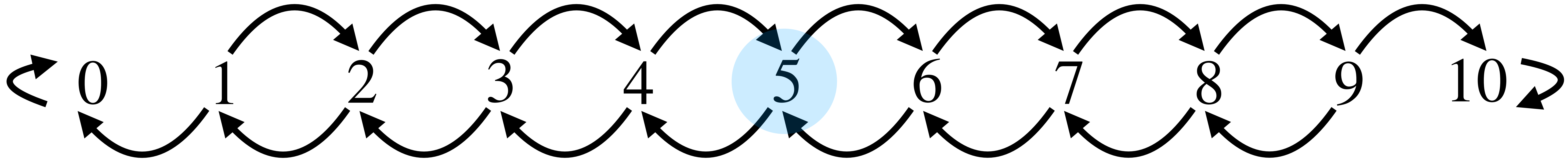
Definition. A **random walk** on an unweighted directed graph G with nodes $\{1,\ldots,n\}$ starting at v is the following process:

- if v has **out-degree** k, roll a k-sided die
- if you rolled an i, go to the ith largest node
- repeat

Warm-Up Example: Gamblers Ruin

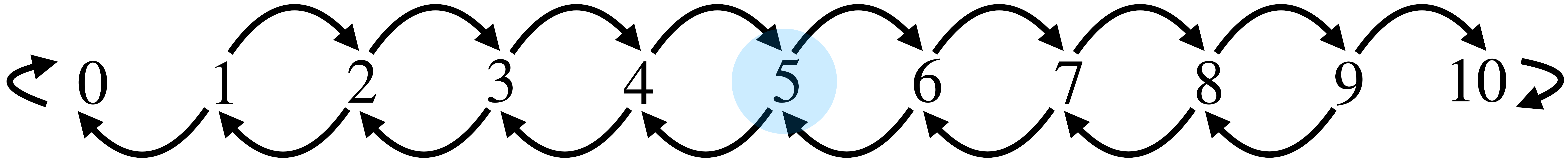


Warm-Up Example: Gamblers Ruin



A single player game:

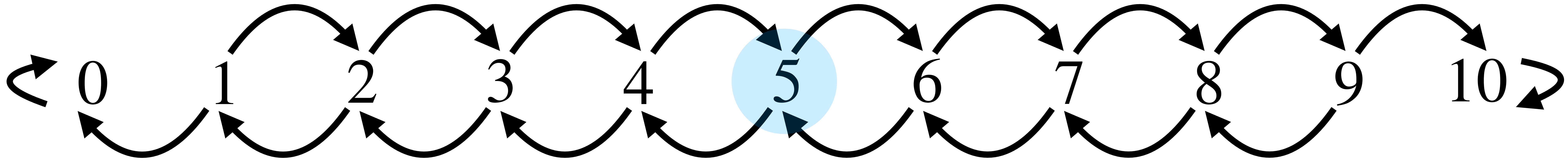
Warm-Up Example: Gamblers Ruin



A single player game:

» The player starts with 5 points.
» They flips a coin.

Warm-Up Example: Gamblers Ruin

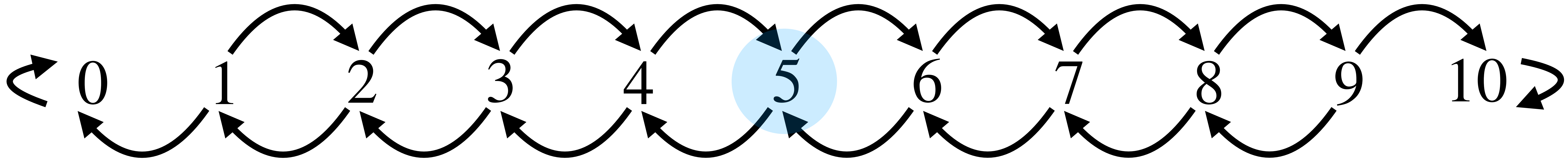


A single player game:

» The player starts with 5 points.
» They flips a coin.

» If its head they get 1 point.
» If its tails they lose 1 point.

Warm-Up Example: Gamblers Ruin



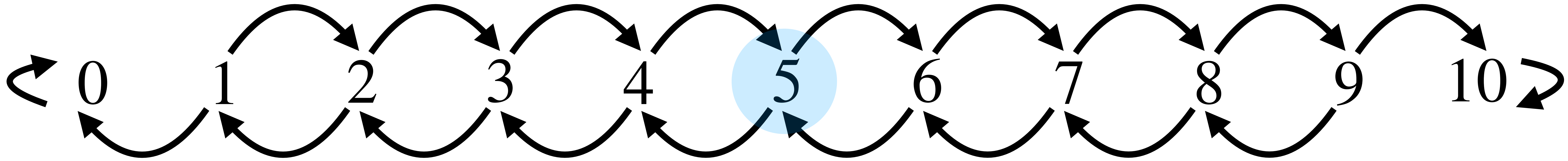
A single player game:

» The player starts with 5 points.
» They flips a coin.

» If its head they get 1 point.
» If its tails they lose 1 point.

» They win if they get to 10 points.
» They lose if they get to 0 points.

Warm-Up Example: Gamblers Ruin



A single player game:

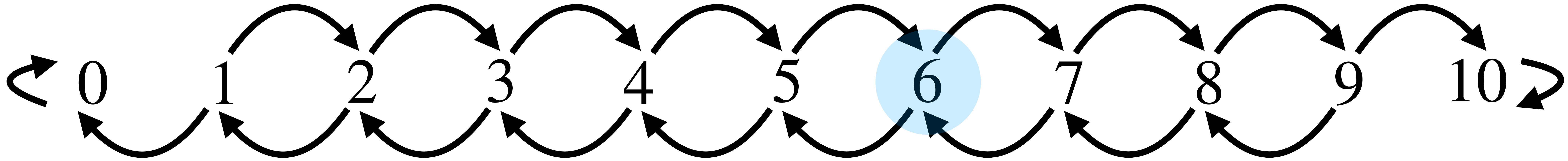
» The player starts with 5 points.
» They flips a coin.

» If its head they get 1 point.
» If its tails they lose 1 point.

» They win if they get to 10 points.
» They lose if they get to 0 points.

We can think of this as a random walk on the above graph

Warm-Up Example: Gamblers Ruin



A single player game:

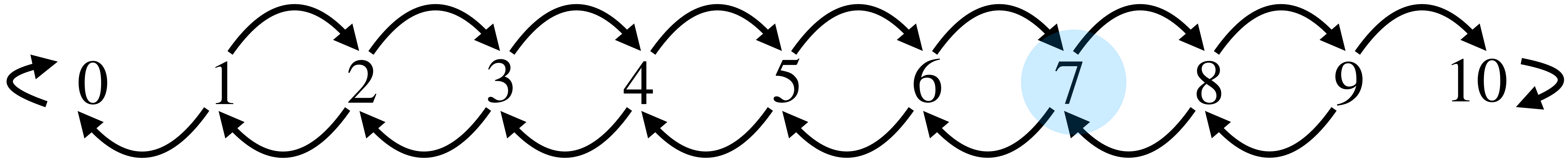
» The player starts with 5 points.
» They flips a coin.

» If its head they get 1 point.
» If its tails they lose 1 point.

» They win if they get to 10 points.
» They lose if they get to 0 points.

We can think of this as a random walk on the above graph

Warm-Up Example: Gamblers Ruin



A single player game:

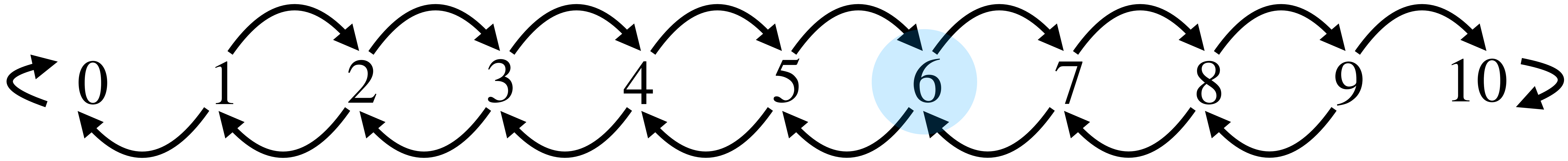
» The player starts with 5 points.
» They flips a coin.

» If its head they get 1 point.
» If its tails they lose 1 point.

» They win if they get to 10 points.
» They lose if they get to 0 points.

We can think of this as a random walk on the above graph

Warm-Up Example: Gamblers Ruin



A single player game:

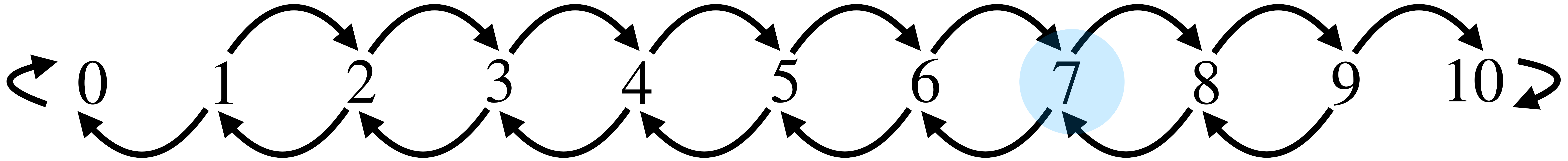
- » The player starts with 5 points.
- » They flips a coin.

- » If its head they get 1 point.
- » If its tails they lose 1 point.

- » They win if they get to 10 points.
- » They lose if they get to 0 points.

We can think of this as a random walk on the above graph

Warm-Up Example: Gamblers Ruin



A single player game:

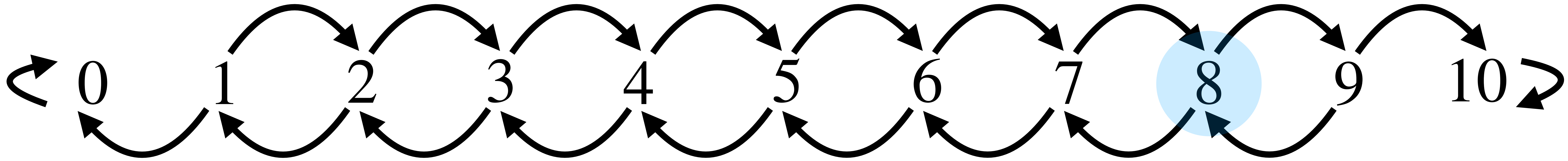
- The player starts with 5 points.
- They flips a coin.

- If its head they get 1 point.
- If its tails they lose 1 point.

- They win if they get to 10 points.
- They lose if they get to 0 points.

We can think of this as a random walk on the above graph

Warm-Up Example: Gamblers Ruin



A single player game:

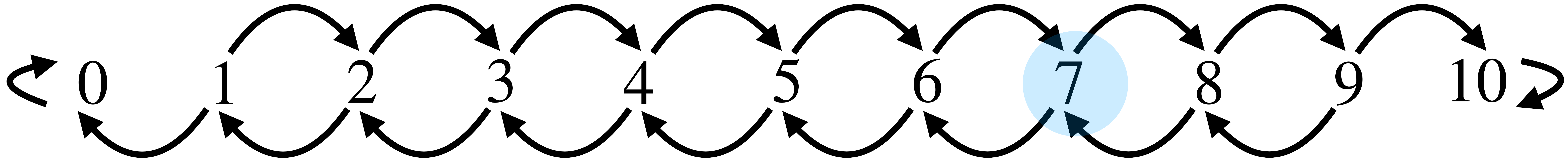
» The player starts with 5 points.
» They flips a coin.

» If its head they get 1 point.
» If its tails they lose 1 point.

» They win if they get to 10 points.
» They lose if they get to 0 points.

We can think of this as a random walk on the above graph

Warm-Up Example: Gamblers Ruin



A single player game:

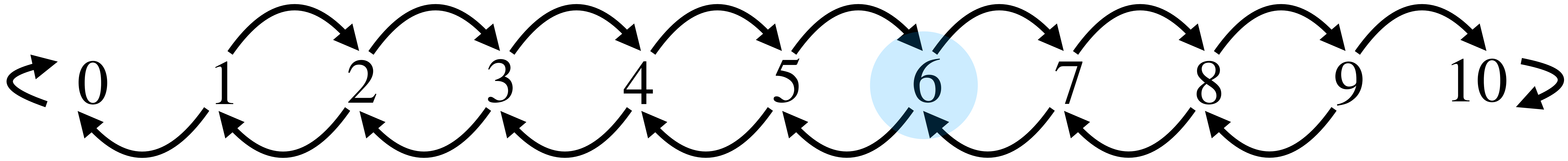
» The player starts with 5 points.
» They flips a coin.

» If its head they get 1 point.
» If its tails they lose 1 point.

» They win if they get to 10 points.
» They lose if they get to 0 points.

We can think of this as a random walk on the above graph

Warm-Up Example: Gamblers Ruin



A single player game:

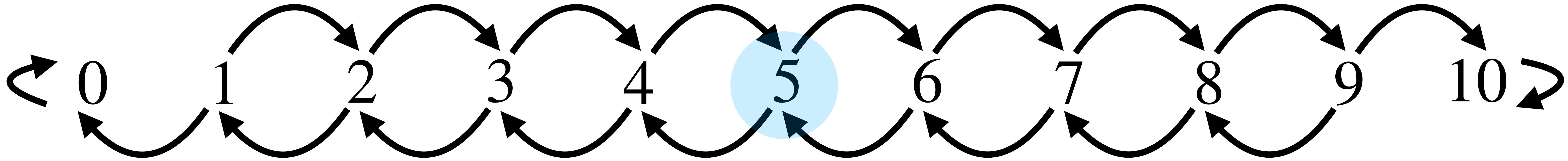
» The player starts with 5 points.
» They flips a coin.

» If its head they get 1 point.
» If its tails they lose 1 point.

» They win if they get to 10 points.
» They lose if they get to 0 points.

We can think of this as a random walk on the above graph

Warm-Up Example: Gamblers Ruin



A single player game:

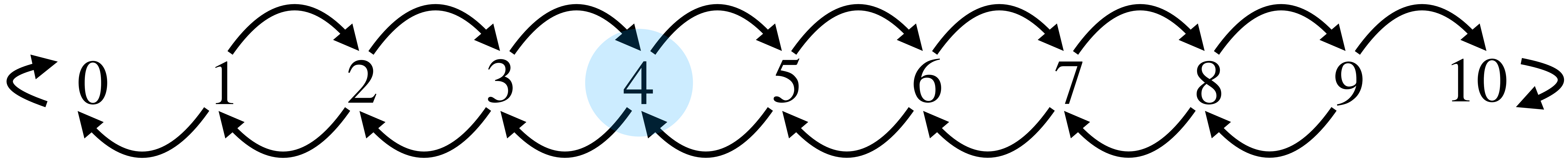
» The player starts with 5 points.
» They flips a coin.

» If its head they get 1 point.
» If its tails they lose 1 point.

» They win if they get to 10 points.
» They lose if they get to 0 points.

We can think of this as a random walk on the above graph

Warm-Up Example: Gamblers Ruin



A single player game:

» The player starts with 5 points.
» They flips a coin.

» If its head they get 1 point.
» If its tails they lose 1 point.

» They win if they get to 10 points.
» They lose if they get to 0 points.

We can think of this as a random walk on the above graph

Normalization and Transition Matrices

$$\begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix} \mapsto \begin{bmatrix} 0 & 0 & 1/3 & 0 & 0 & 0 \\ 1/2 & 0 & 1/3 & 0 & 0 & 0 \\ 1/2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1/2 & 1 \\ 0 & 0 & 1/3 & 1/2 & 0 & 0 \\ 0 & 0 & 0 & 1/2 & 1/2 & 0 \end{bmatrix}$$

Adjacency Matrix **Transition Matrix**

Normalization is the process of preprocessing an adjacency matrix so that (almost) every column <u>sums to 1</u>.

Normalization and Transition Matrices

$$\underbrace{\begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}}_{\texttt{Adjacency Matrix}} \mapsto \underbrace{\begin{bmatrix} 0 & 0 & 1/3 & 0 & 0 & 0 \\ 1/2 & 0 & \overset{\Pr(\texttt{going from } 3 \to 2)}{\boxed{1/3}} & 0 & 0 & 0 \\ 1/2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1/2 & 1 \\ 0 & 0 & 1/3 & 1/2 & 0 & 0 \\ 0 & 0 & 0 & 1/2 & 1/2 & 0 \end{bmatrix}}_{\textbf{Transition Matrix}}$$

Normalization is the process of preprocessing an adjacency matrix so that (almost) every column <u>sums to 1</u>.

Recall: Stochastic Matrices

Definition. A $n \times n$ matrix is **stochastic** if its entries are nonnegative and its columns sum to 1.

Example.

$$\begin{bmatrix} 0.7 & 0.1 & 0.3 \\ 0.2 & 0.8 & 0.3 \\ 0.1 & 0.1 & 0.4 \end{bmatrix}$$

Recall: Markov Chains

Definition. A **Markov chain** is a linear dynamical system whose evolution function is given by a <u>stochastic</u> matrix.

(We can construct a "chain" of state vectors, where each state vector only depends on the one before it.)

So we can consider the **Markov Chain** associated with a random walk

We did this in Homework 6

```python
def adjacency_to_stochastic(a):
    for i in range(a.shape[0]):
        div = np.sum(a[:,i])
        if div != 0:
            a[:,i] /= div
```

```python
def random_step(a, i):
    rng = np.random.default_rng()
    return rng.choice(a.shape[0], p=a[:, i])
```

```python
def random_walk(a, i, length):
    walk = []
    next_index = i
    for _ in range(length):
        next_index = random_step(a, next_index)
        walk.append(next_index)
```

Recall: Steady-State Vectors

Definition. A **steady-state vector** for a stochastic matrix A is a probability vector $\mathbf{q}$ such that

$$A\mathbf{q} = \mathbf{q}$$

A steady-state vector is *not changed* by the stochastic matrix. They describe equilibrium distributions.

Recall: Steady-State Vectors

Definition. A **steady-state vector** for a stochastic matrix A is a probability vector $\mathbf{q}$ s

A steady state of A is an eigenvector with eigenvalue 1.

A steady-state vector is *not changed* by the stochastic matrix. They describe <u>equilibrium distributions</u>.

How do we interpret steady states of random walks?

Recall: Steady States of Random Walks

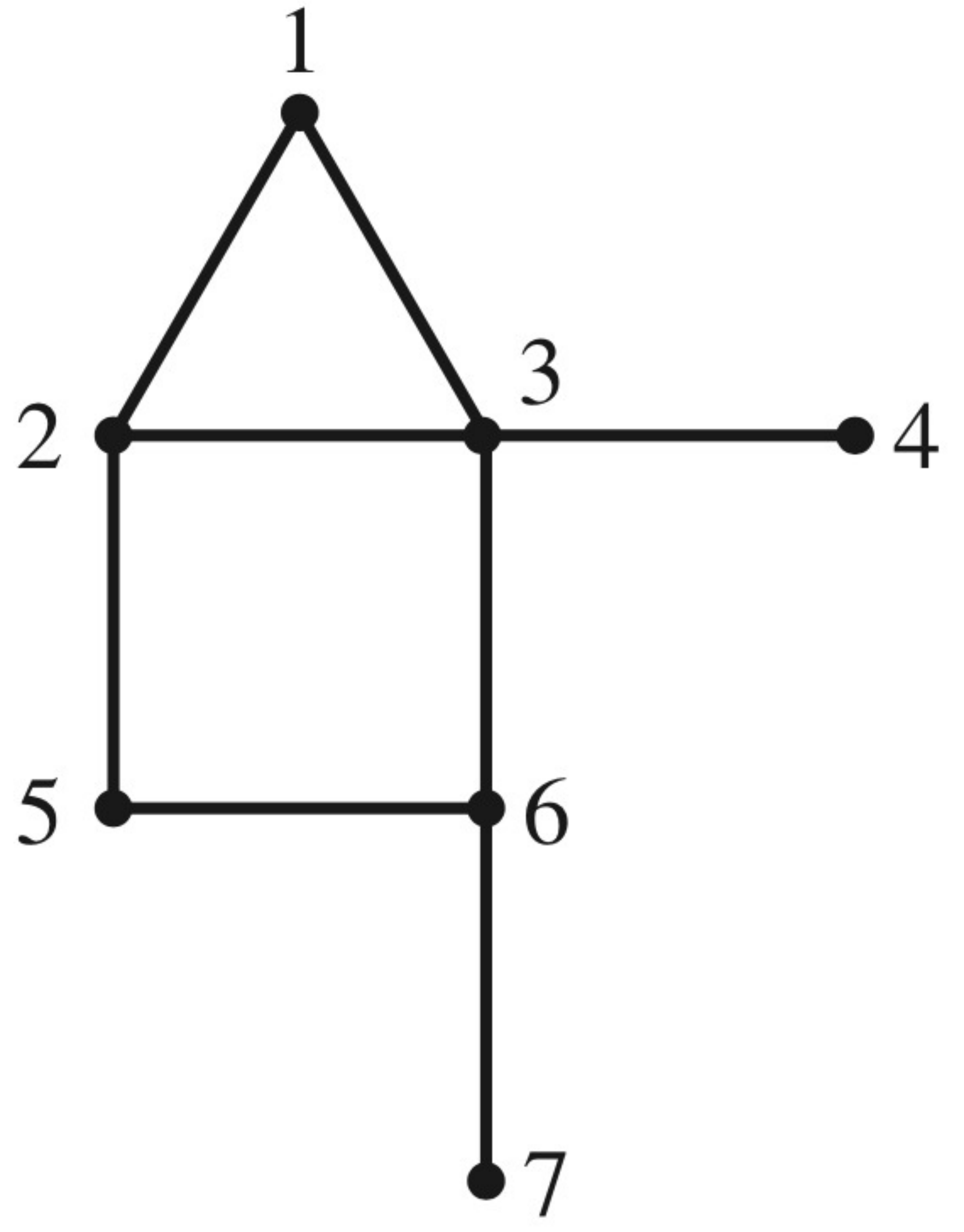
`If a random walk goes on for a sufficiently long time, then the probability that we end up in a particular place becomes fixed.`

`If you wander for a sufficiently long time, it doesn't matter where you started.`

Fundamental Question

How do we (quickly) determine a steady state of a random walk?

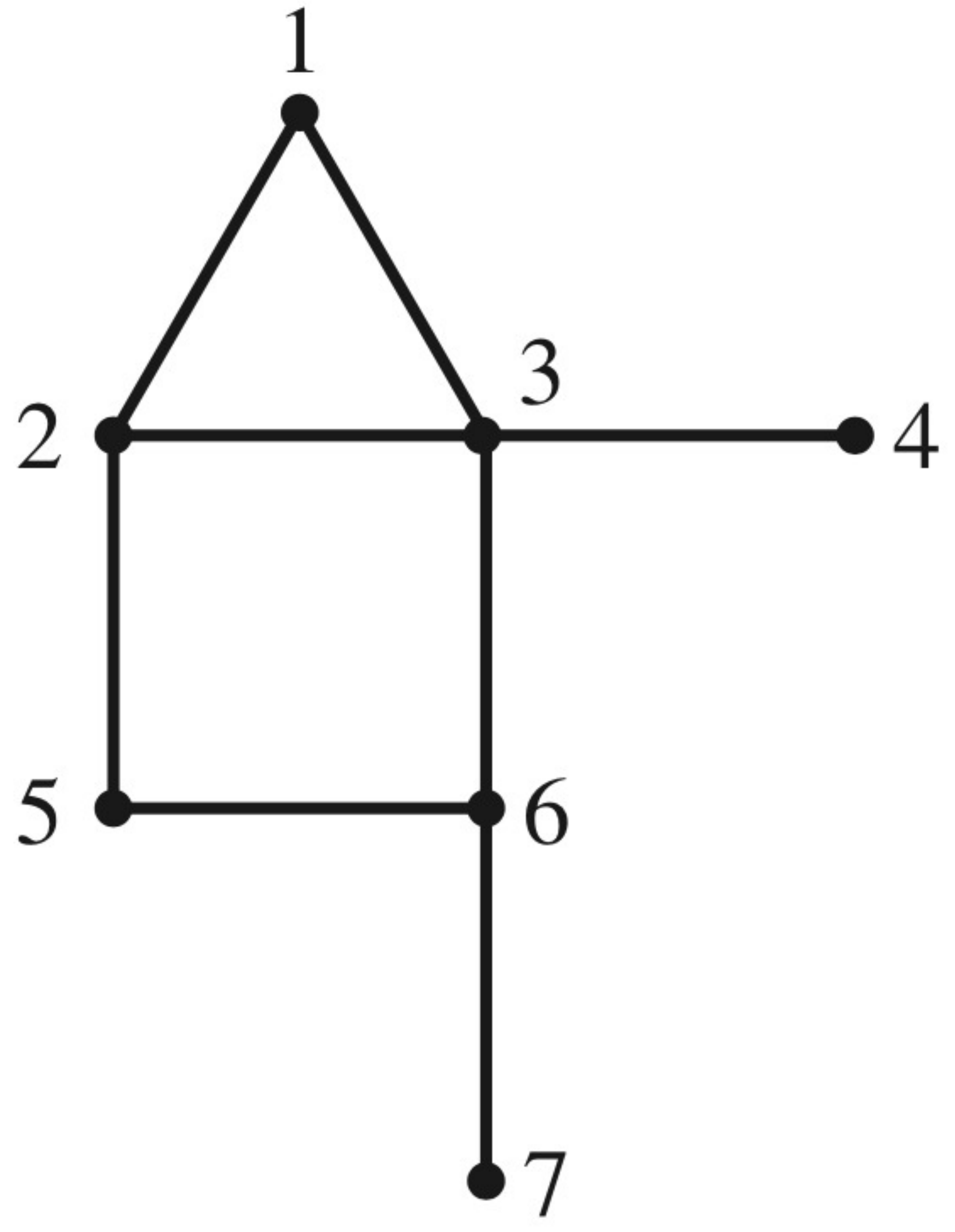
Special Case: Undirected Graphs



Linear Algebra and its Applications, Lay, Lay, McDonald

Special Case: Undirected Graphs

Note. *An undirected graph is just a directed in which both directions of edges are always present.*

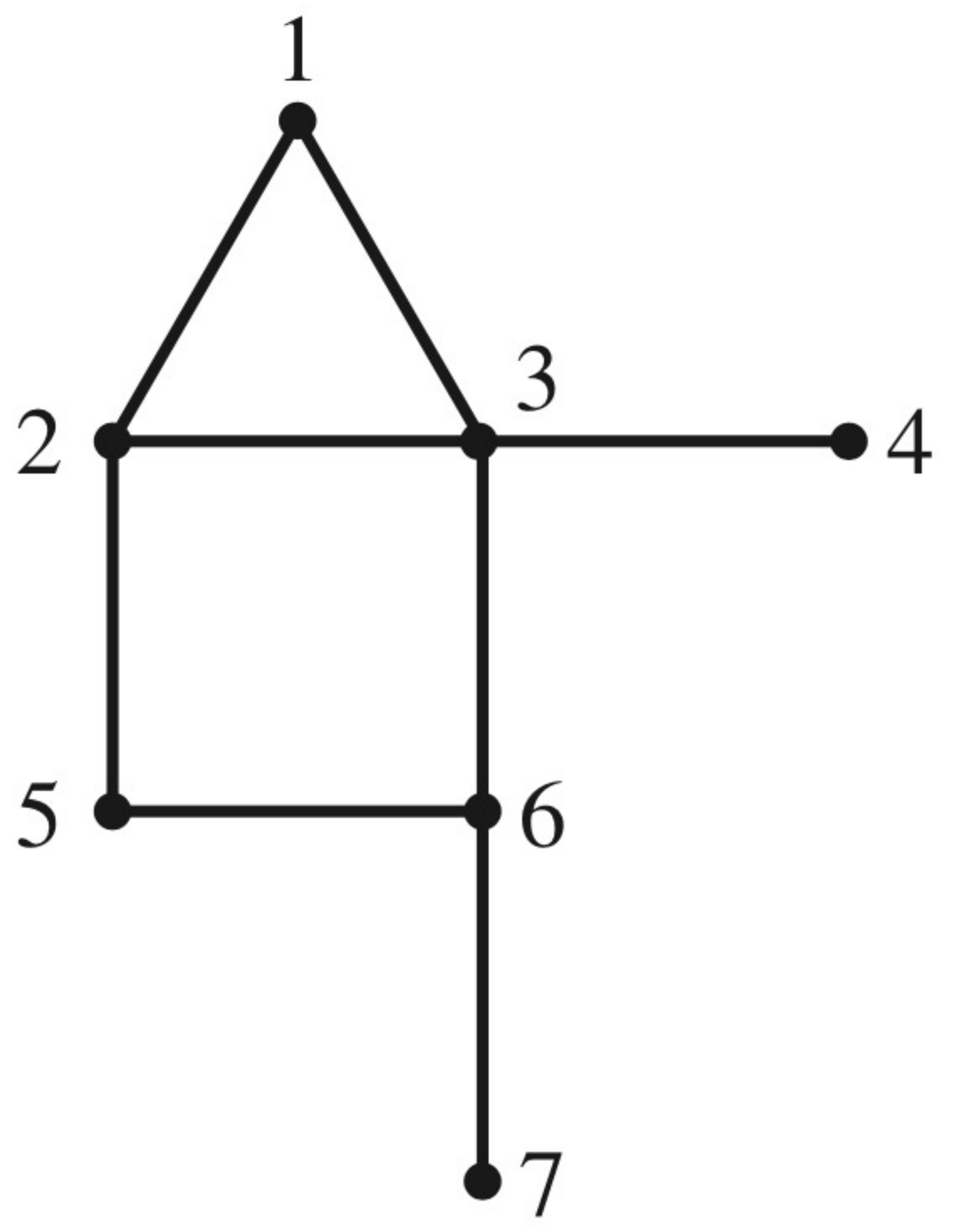


Special Case: Undirected Graphs

Note. *An undirected graph is just a directed in which both directions of edges are always present.*

Theorem. The steady state vector of a random walk on an undirected graph is

$$\frac{1}{\sum_{i=1}^{n} \deg(i)} \begin{bmatrix} \deg(1) \\ \deg(2) \\ \vdots \\ \deg(n) \end{bmatrix}$$



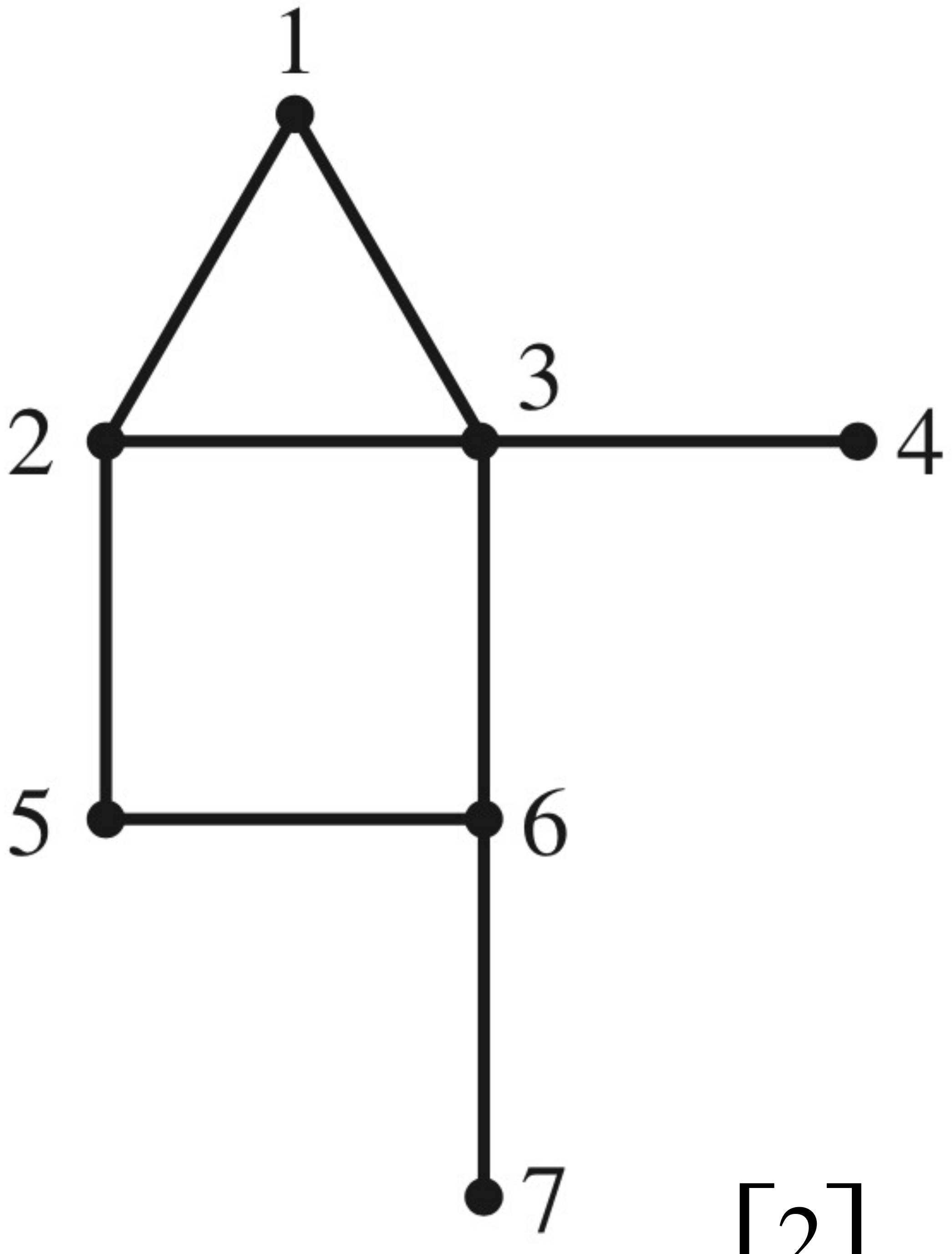
Linear Algebra and its Applications, Lay, Lay, McDonald

Special Case: Undirected Graphs

Note. *An undirected graph is just a directed in which both directions of edges are always present.*

Theorem. The steady state vector of a random walk on an undirected graph is

$$\frac{1}{\sum_{i=1}^{n} \deg(i)} \begin{bmatrix} \deg(1) \\ \deg(2) \\ \vdots \\ \deg(n) \end{bmatrix}$$



$$\text{steadyState} = \frac{1}{16} \begin{bmatrix} 2 \\ 3 \\ 4 \\ 1 \\ 2 \\ 3 \\ 1 \end{bmatrix}$$

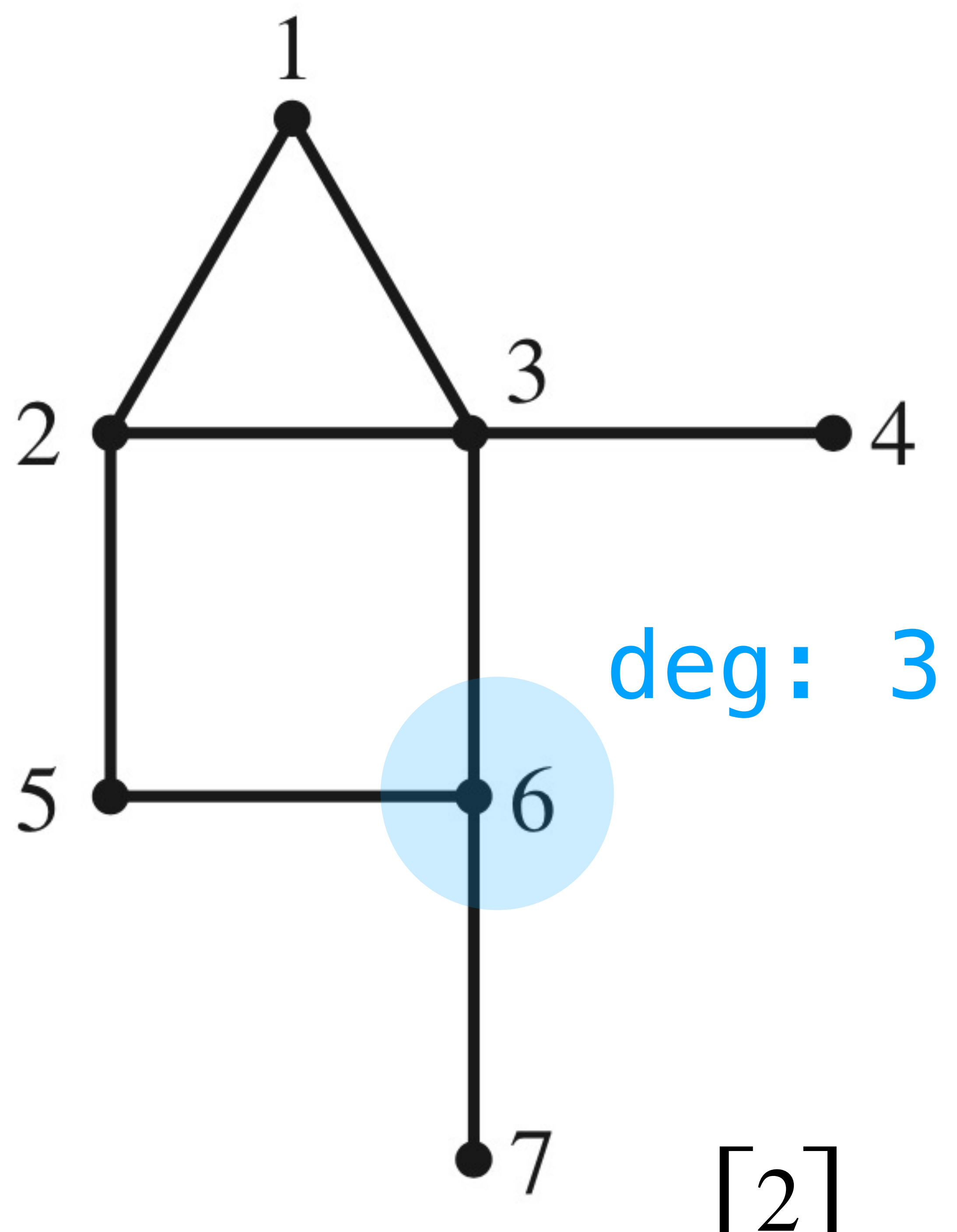
Linear Algebra and its Applications, Lay, Lay, McDonald

Special Case: Undirected Graphs

Note. *An undirected graph is just a directed in which both directions of edges are always present.*

Theorem. The steady state vector of a random walk on an undirected graph is

$$\frac{1}{\sum_{i=1}^{n} \deg(i)} \begin{bmatrix} \deg(1) \\ \deg(2) \\ \vdots \\ \deg(n) \end{bmatrix}$$



$$\text{steadyState} = \frac{1}{16} \begin{bmatrix} 2 \\ 3 \\ 4 \\ 1 \\ 2 \\ 3 \\ 1 \end{bmatrix}$$

Linear Algebra and its Applications, Lay, Lay, McDonald

Special Case: Undirected Graphs

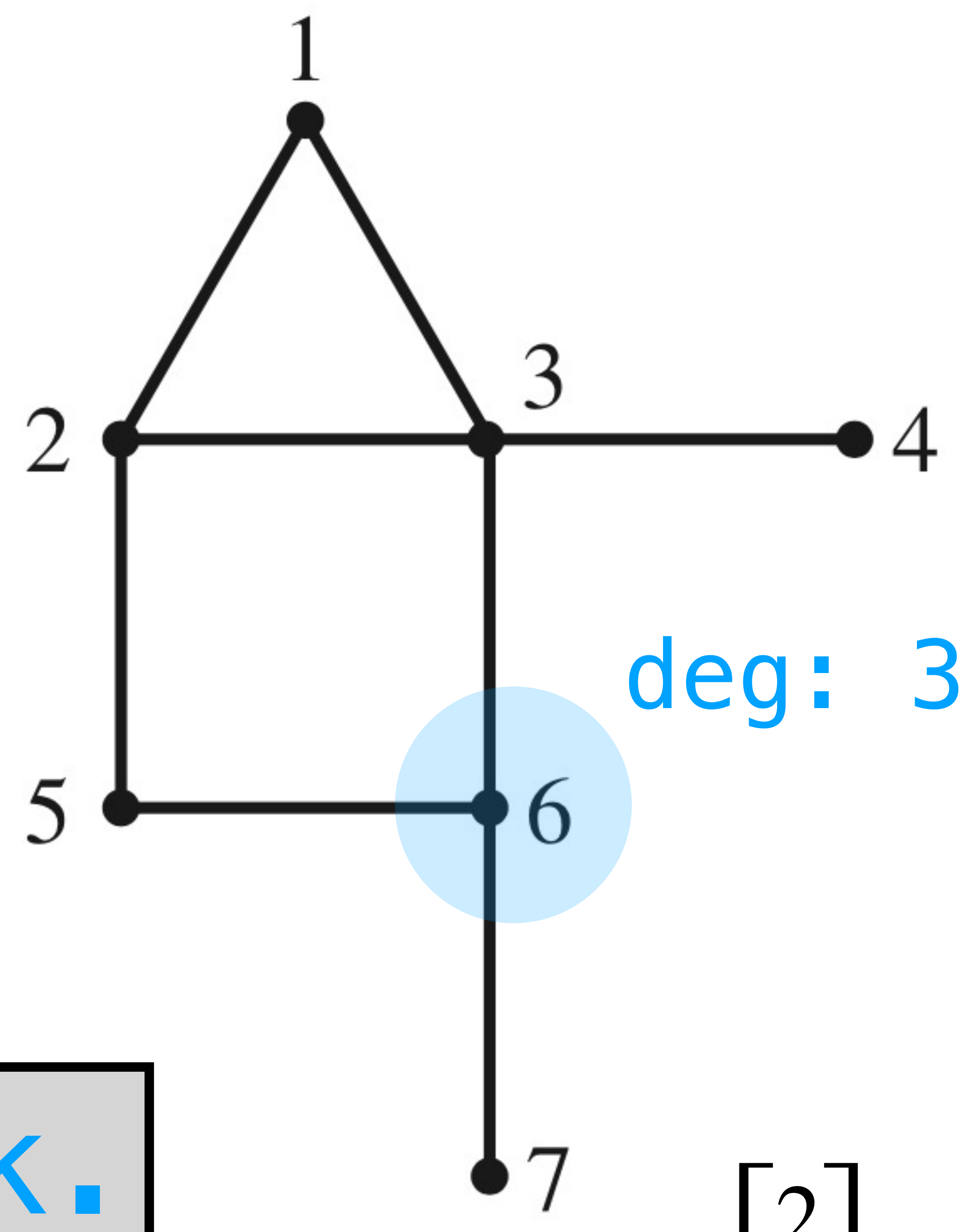
Note. *An undirected graph is just a directed in which both directions of edges are always present.*

Theorem. The steady state vector of a random w

We don't need to do any work.

graph is

$$\frac{1}{\sum_{i=1}^{n} \deg(i)} \begin{bmatrix} \deg(1) \\ \deg(2) \\ \vdots \\ \deg(n) \end{bmatrix}$$



$$\text{steadyState} = \frac{1}{16} \begin{bmatrix} 2 \\ 3 \\ 4 \\ 1 \\ 2 \\ 3 \\ 1 \end{bmatrix}$$

Linear Algebra and its Applications, Lay, Lay, McDonald

The Random Surfer Model

The random surfer is not on an undirected graph

2.1.2. Intuitive justification

PageRank can be thought of as a model of user behavior. We assume there is a "random surfer" who is given a Web page at random and keeps clicking on links, never hitting "back" but eventually gets bored and starts on another random page. The probability that the random surfer visits a page is its PageRank

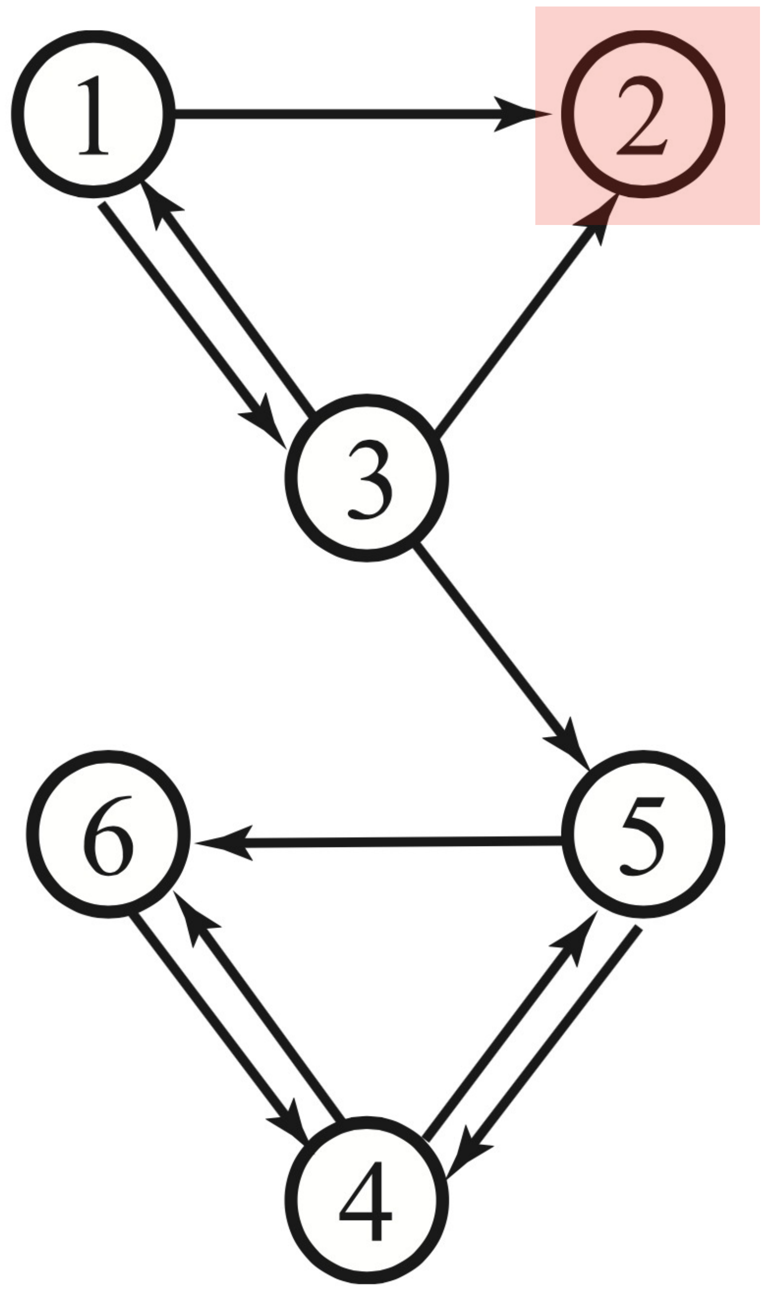
The anatomy of a large-scale hypertextual Web search engine (1998)

PageRank requires quickly finding steady-states for directed graphs

Tricky Issue: Boundaries

$$\begin{bmatrix} 0 & 0 & 1/3 & 0 & 0 & 0 \\ 1/2 & 0 & 1/3 & 0 & 0 & 0 \\ 1/2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1/2 & 1 \\ 0 & 0 & 1/3 & 1/2 & 0 & 0 \\ 0 & 0 & 0 & 1/2 & 1/2 & 0 \end{bmatrix}$$

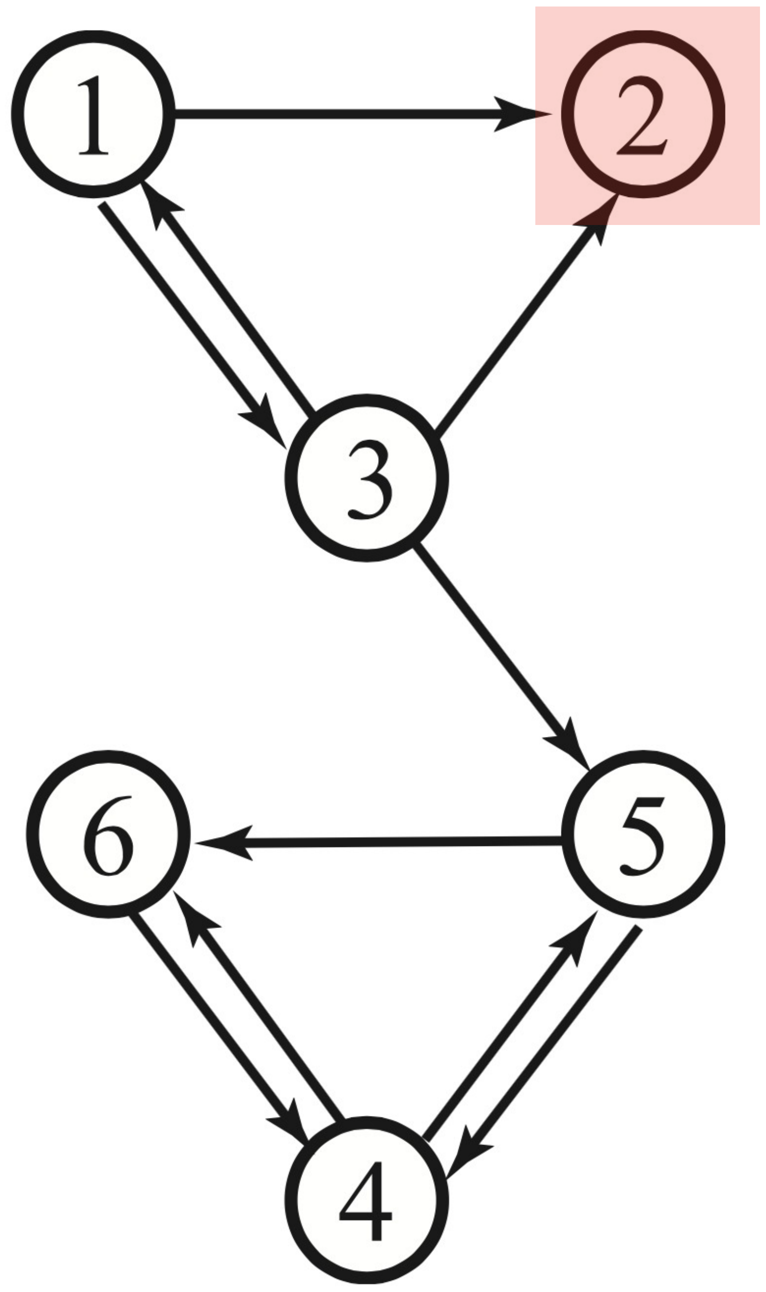
There is no way to leave (2)



Tricky Issue: Boundaries

$$\begin{bmatrix} 0 & 0 & 1/3 & 0 & 0 & 0 \\ 1/2 & 0 & 1/3 & 0 & 0 & 0 \\ 1/2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1/2 & 1 \\ 0 & 0 & 1/3 & 1/2 & 0 & 0 \\ 0 & 0 & 0 & 1/2 & 1/2 & 0 \end{bmatrix}$$

There is no way to leave (2)

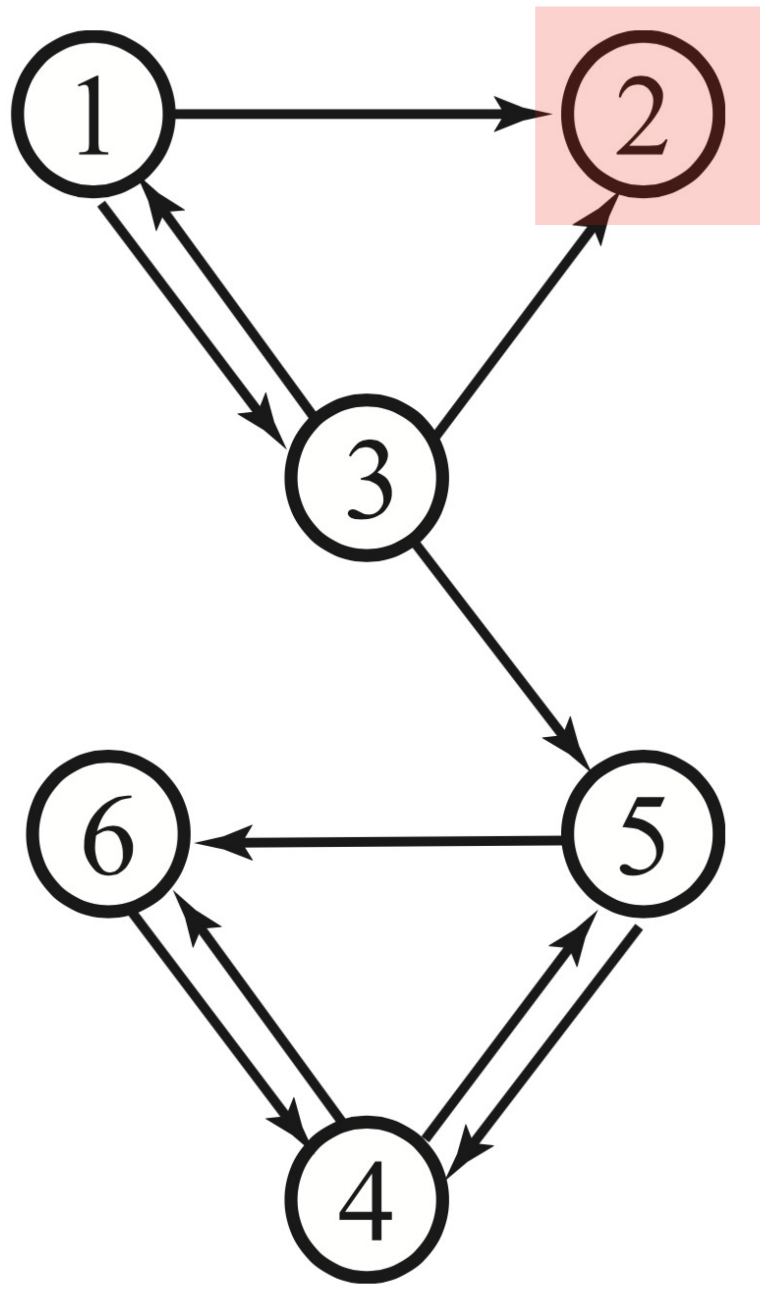


The transition matrix of a graph may not actually be stochastic because of 0s columns.

Tricky Issue: Boundaries

There is no way to leave (2)

$$\begin{bmatrix} 0 & 0 & 1/3 & 0 & 0 & 0 \\ 1/2 & 0 & 1/3 & 0 & 0 & 0 \\ 1/2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1/2 & 1 \\ 0 & 0 & 1/3 & 1/2 & 0 & 0 \\ 0 & 0 & 0 & 1/2 & 1/2 & 0 \end{bmatrix}$$



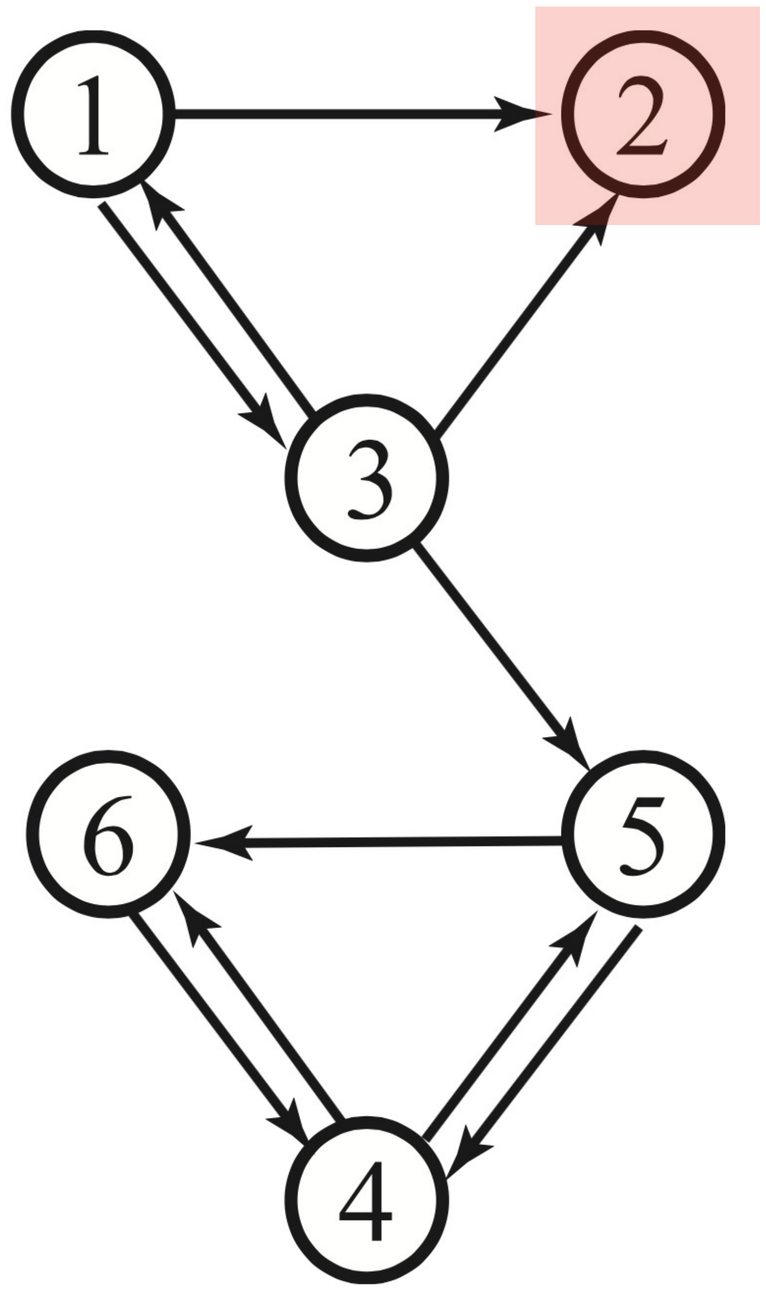
The transition matrix of a graph may not actually be stochastic because of 0s columns.

We can't use standard techniques for Markov Chains.

Tricky Issue: Boundaries

$$\begin{bmatrix} 0 & 0 & 1/3 & 0 & 0 & 0 \\ 1/2 & 0 & 1/3 & 0 & 0 & 0 \\ 1/2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1/2 & 1 \\ 0 & 0 & 1/3 & 1/2 & 0 & 0 \\ 0 & 0 & 0 & 1/2 & 1/2 & 0 \end{bmatrix}$$

There is no way to leave (2)



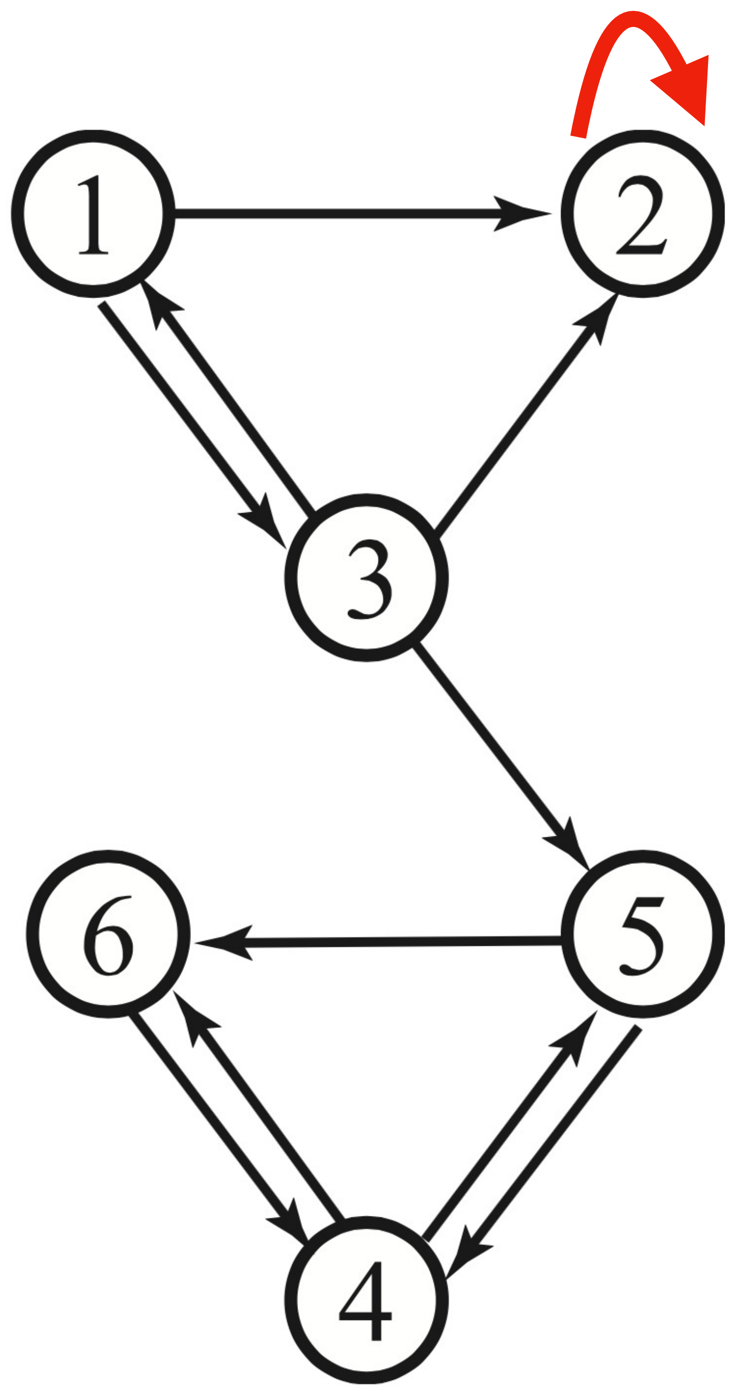
The transition matrix of a graph may not actually be stochastic because of 0s columns.

We can't use standard techniques for Markov Chains.

There are two typical fixes to this.

Absorbing Boundaries

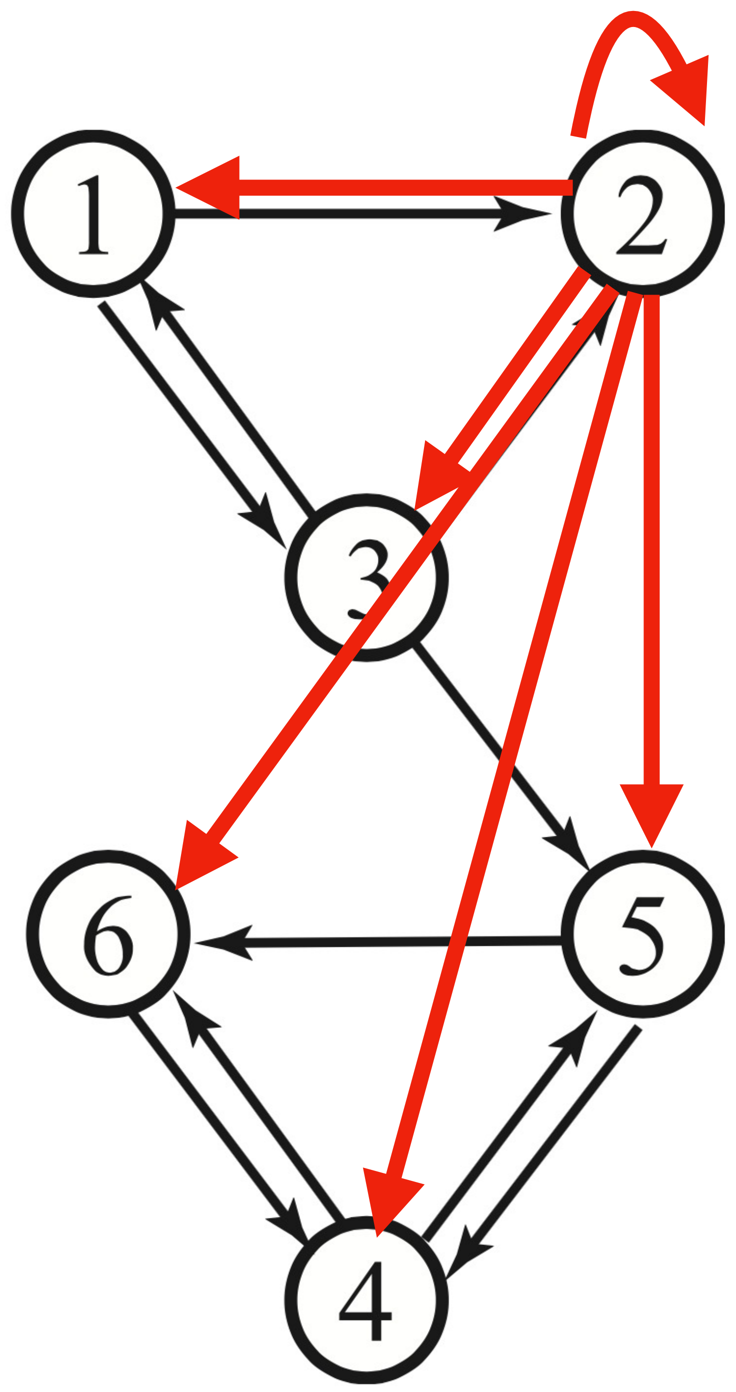
$$\begin{bmatrix} 0 & 0 & 1/3 & 0 & 0 & 0 \\ 1/2 & \boxed{1} & 1/3 & 0 & 0 & 0 \\ 1/2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1/2 & 1 \\ 0 & 0 & 1/3 & 1/2 & 0 & 0 \\ 0 & 0 & 0 & 1/2 & 1/2 & 0 \end{bmatrix}$$



We create a *self-loop* at the boundaries so that we stay at the node when we get there.

Reflecting Boundaries

$$\begin{bmatrix} 0 & 1/6 & 1/3 & 0 & 0 & 0 \\ 1/2 & 1/6 & 1/3 & 0 & 0 & 0 \\ 1/2 & 1/6 & 0 & 0 & 0 & 0 \\ 0 & 1/6 & 0 & 0 & 1/2 & 1 \\ 0 & 1/6 & 1/3 & 1/2 & 0 & 0 \\ 0 & 1/6 & 0 & 1/2 & 1/2 & 0 \end{bmatrix}$$



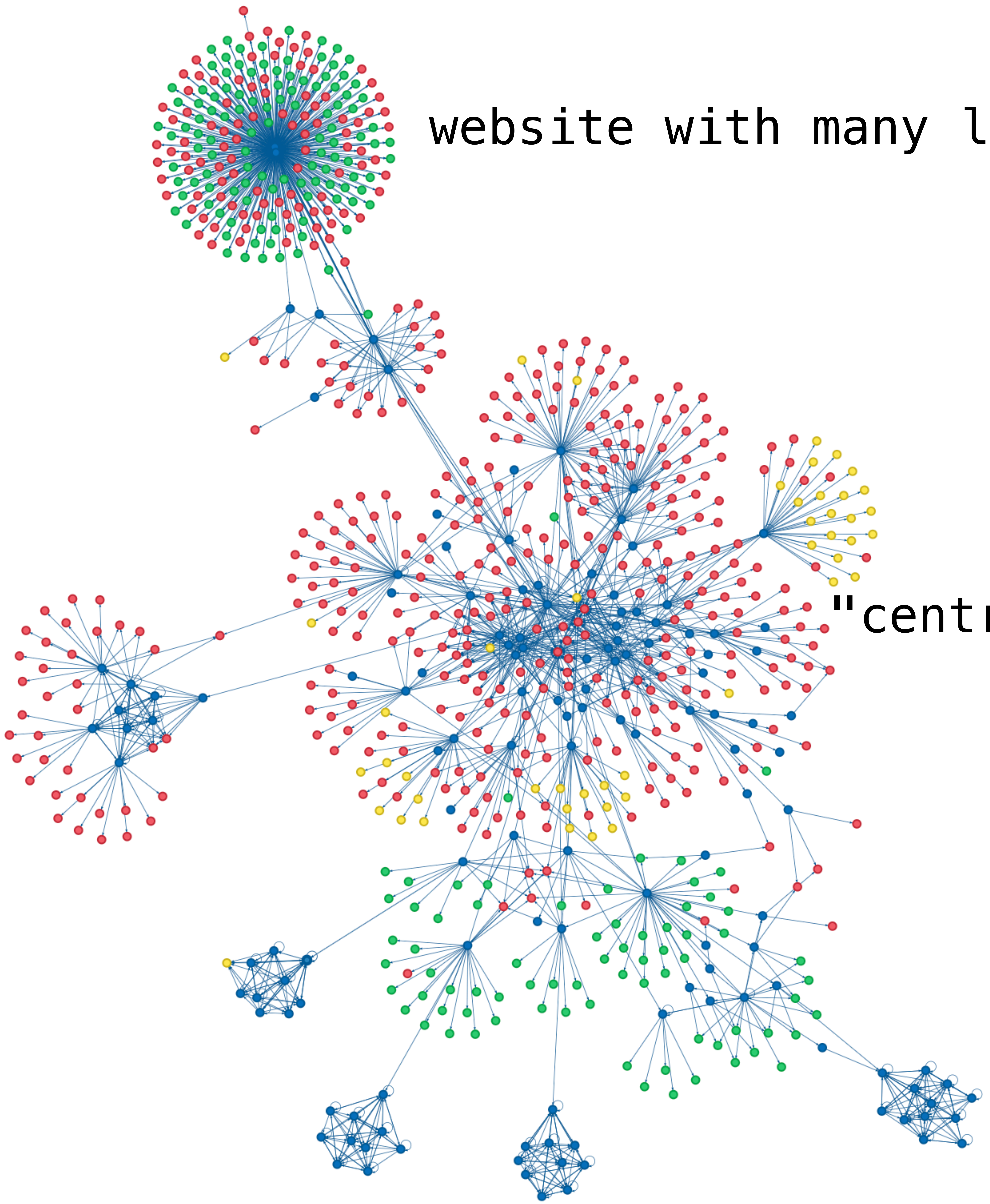
We make it possible to go *anywhere* after getting to a boundary.

Moving Forward

What is the connection between steady states and website importance?

PageRank

The Picture



website with many links

"central" website linked to many times

Page Importance

$$\mathsf{Importance}(k) = \sum_j \mathsf{Pr}(\texttt{going from } j \to k) \cdot \mathsf{Importance}(j)$$

Page Importance

$$\mathsf{Importance}(k) = \sum_j \mathsf{Pr}(\texttt{going from } j \to k) \cdot \mathsf{Importance}(j)$$

We're interested in defining a function $\mathsf{Importance}(\cdot)$ which tells us how important website k is.

Page Importance

$$\text{Importance}(k) = \sum_j \text{Pr}(\texttt{going from } j \to k) \cdot \text{Importance}(j)$$

We're interested in defining a function $\text{Importance}(\cdot)$ which tells us how important website k is.

A website is important if it is linked to by many important websites.

Page Importance

$$\text{Importance}(k) = \sum_j \text{Pr}(\texttt{going from } j \to k) \cdot \text{Importance}(j)$$

We're interested in defining a function $\text{Importance}(\cdot)$ which tells us how important website k is.

A website is important if it is linked to by many important websites.

This is circular, but familiar...

Importance Vector

$$\mathtt{Importance}_k = \sum_{i=1}^{n} \mathsf{Pr}(\mathtt{going\ from}\ j \to k) \cdot \mathtt{Importance}_\mathsf{j}$$

Importance Vector

$$\text{Importance}_k = \sum_{i=1}^{n} \text{Pr}(\texttt{going from } j \to k) \cdot \text{Importance}_{\text{j}}$$

Instead, let's say we're trying to find an *importance vector*, whose kth component is the importance of website k.

Importance Vector

$$\text{Importance}_k = \sum_{i=1}^{n} \Pr(\texttt{going from } j \to k) \cdot \text{Importance}_\text{j}$$

Instead, let's say we're trying to find an *importance vector*, whose kth component is the importance of website k.

Then we recognize that these probabilities are entries of a transition matrix...

Importance Vector

$$\text{Importance}_k = \sum_{i=1}^{n} A_{kj} \cdot \text{Importance}_j$$
$$= (A \cdot \text{Importance})_k$$

Importance Vector

$$\text{Importance}_k = \sum_{i=1}^{n} A_{kj} \cdot \text{Importance}_j$$
$$= (A \cdot \text{Importance})_k$$

where A is a transition matrix for the part of the web associate with our search term.

Importance Vector

$$\text{Importance}_k = \sum_{i=1}^{n} A_{kj} \cdot \text{Importance}_j$$
$$= (A \cdot \text{Importance})_k$$

where A is a transition matrix for the part of the web associate with our search term.

$$A \cdot \text{Importance} = \text{Importance}$$

Importance Vector

$$\text{Importance}_k = \sum_{i=1}^{n} A_{kj} \cdot \text{Importance}_j$$
$$= (A \cdot \text{Importance})_k$$

where A is a transition matrix for the part of the web associate with our search term.

$$A \cdot \text{Importance} = \text{Importance}$$

The importance vectors is a steady state.

Page Importance and Eigenvectors

$$A \cdot \text{Importance} = \text{Importance}$$

Page Importance and Eigenvectors

$$A \cdot \underbrace{\text{Importance}}_{\text{eigenvector}} = \text{Importance}$$

The eigenvector with eigenvalue 1 of our transition matrix is our *importance vector*.

Page Importance and Eigenvectors

$$A \cdot \boxed{\text{Importance}} = \boxed{\text{Importance}}$$

eigenvector

The eigenvector with eigenvalue 1 of our transition matrix is our *importance vector*.

We order webpages by importance, so this gives a **ranking** of webpages.

Page Importance and Eigenvectors

$$A \cdot \boxed{\text{Importance}} = \boxed{\text{Importance}}$$

eigenvector

The eigenvector with eigenvalue 1 of our transition matrix is our *importance vector*.

We order webpages by importance, so this gives a **ranking** of webpages.

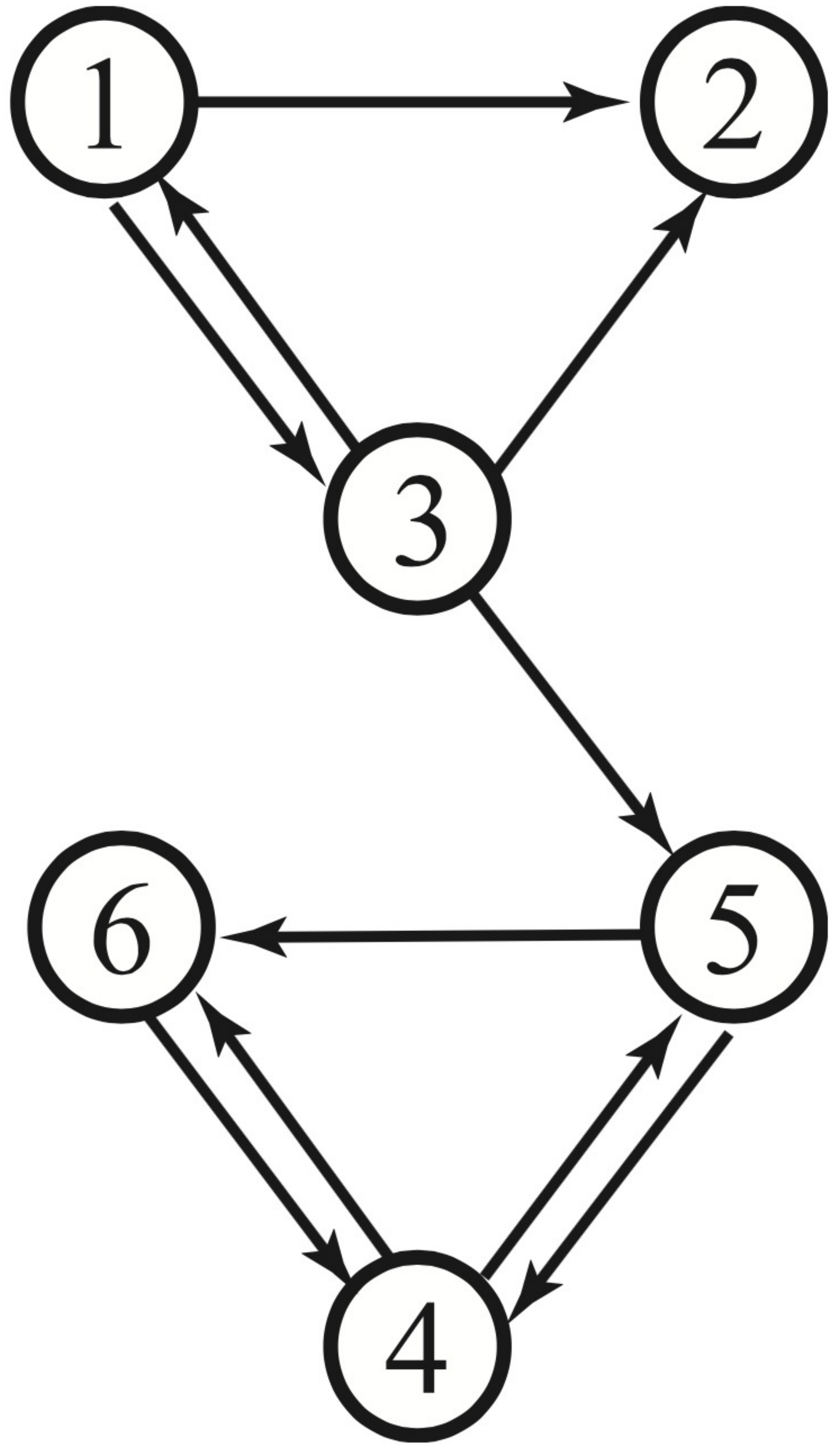
This vector tells us the probability a random surfer is on a given page **in the long term.**

The Algorithm

The Algorithm (High Level)

PageRank

1. Build a graph encoding the websites and their links for the query we're given.
2. Build the adjacency matrix for this graph.
3. Turn boundaries into reflectors.
4. Normalize the matrix.
5. *Add a damping factor.*
6. Compute the eigenvector for the largest eigenvalues.
7. Order indices according to the entries of this vector.



The Algorithm (High Level)

PageRank

1. Build a graph encoding the websites and their links for the query we're given.
2. Build the adjacency matrix for this graph.
3. Turn boundaries into reflectors.
4. Normalize the matrix.
5. *Add a damping factor.*
6. Compute the eigenvector for the largest eigenvalues.
7. Order indices according to the entries of this vector.

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

The Algorithm (High Level)

PageRank

1. Build a graph encoding the websites and their links for the query we're given.
2. Build the adjacency matrix for this graph.
3. Turn boundaries into reflectors.
4. Normalize the matrix.
5. *Add a damping factor.*
6. Compute the eigenvector for the largest eigenvalues.
7. Order indices according to the entries of this vector.

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

The Algorithm (High Level)

PageRank

1. Build a graph encoding the websites and their links for the query we're given.
2. Build the adjacency matrix for this graph.
3. Turn boundaries into reflectors.
4. Normalize the matrix.
5. *Add a damping factor.*
6. Compute the eigenvector for the largest eigenvalues.
7. Order indices according to the entries of this vector.

$$\begin{bmatrix} 0 & 1/6 & 1/3 & 0 & 0 & 0 \\ 1/2 & 1/6 & 1/3 & 0 & 0 & 0 \\ 1/2 & 1/6 & 0 & 0 & 0 & 0 \\ 0 & 1/6 & 0 & 0 & 1/2 & 1 \\ 0 & 1/6 & 1/3 & 1/2 & 0 & 0 \\ 0 & 1/6 & 0 & 1/2 & 1/2 & 0 \end{bmatrix}$$

The Algorithm (High Level)

PageRank

1. Build a graph encoding the websites and their links for the query we're given.
2. Build the adjacency matrix for this graph.
3. Turn boundaries into reflectors.
4. Normalize the matrix.
5. *Add a damping factor.*
6. Compute the eigenvector for the largest eigenvalues.
7. Order indices according to the entries of this vector.

$$(1-\alpha)\begin{bmatrix} 0 & 1/6 & 1/3 & 0 & 0 & 0 \\ 1/2 & 1/6 & 1/3 & 0 & 0 & 0 \\ 1/2 & 1/6 & 0 & 0 & 0 & 0 \\ 0 & 1/6 & 0 & 0 & 1/2 & 1 \\ 0 & 1/6 & 1/3 & 1/2 & 0 & 0 \\ 0 & 1/6 & 0 & 1/2 & 1/2 & 0 \end{bmatrix} + \frac{\alpha}{6}\mathbf{1}$$

(more on this later)

The Algorithm (High Level)

PageRank

1. Build a graph encoding the websites and their links for the query we're given.
2. Build the adjacency matrix for this graph.
3. Turn boundaries into reflectors.
4. Normalize the matrix.
5. *Add a damping factor.*
6. Compute the eigenvector for the largest eigenvalues.
7. Order indices according to the entries of this vector.

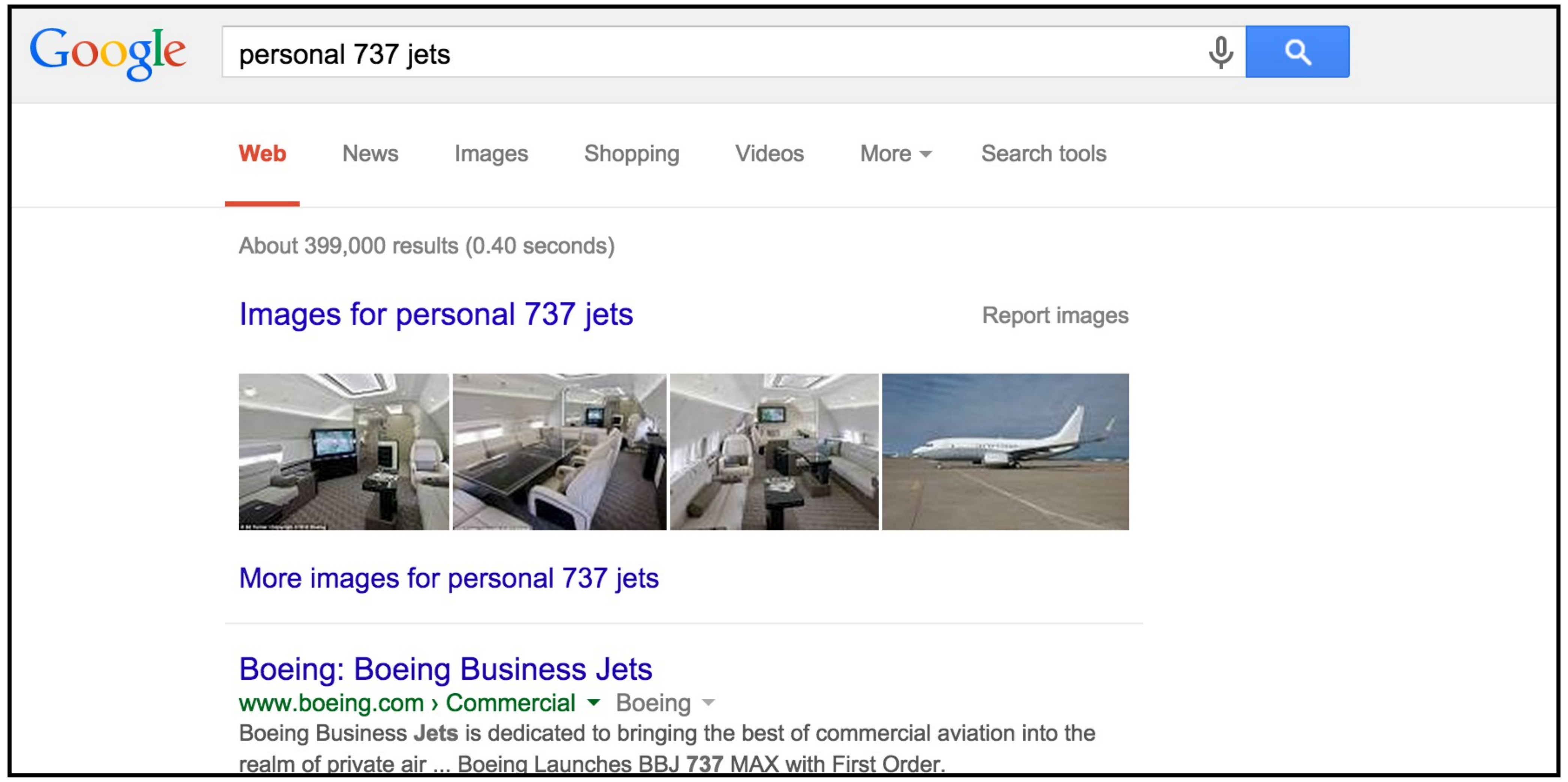
np.linalg.eig(a)

(more on this later)

The Algorithm (High Level)

PageRank

1. Build a graph encoding the websites and their links for the query we're given.
2. Build the adjacency matrix for this graph.
3. Turn boundaries into reflectors.
4. Normalize the matrix.
5. *Add a damping factor.*
6. Compute the eigenvector for the largest eigenvalues.
7. Order indices according to the entries of this vector.



The Algorithm (High Level)

PageRank

1. Build a graph encoding the websites and their links for the query we're given. ✓
2. Build the adjacency matrix for this graph. ✓
3. Turn boundaries into reflectors. ✓
4. Normalize the matrix. ✓
5. *Add a damping factor.*
6. Compute the eigenvector for the largest eigenvalues.
7. Order indices according to the entries of this vector.

We just talked about the importance of these steps.

Damping Factor

PageRank

1. Build a graph encoding the websites and their links for the query we're given.
2. Build the adjacency matrix for this graph.
3. Turn boundaries into reflectors.
4. Normalize the matrix.
5. *Add a damping factor.*
6. Compute the eigenvector for the largest eigenvalues.
7. Order indices according to the entries of this vector.

$$(1-\alpha)\begin{bmatrix} 0 & 1/6 & 1/3 & 0 & 0 & 0 \\ 1/2 & 1/6 & 1/3 & 0 & 0 & 0 \\ 1/2 & 1/6 & 0 & 0 & 0 & 0 \\ 0 & 1/6 & 0 & 0 & 1/2 & 1 \\ 0 & 1/6 & 1/3 & 1/2 & 0 & 0 \\ 0 & 1/6 & 0 & 1/2 & 1/2 & 0 \end{bmatrix} + \frac{\alpha}{6}\mathbf{1}$$

damping factor

Damping Factor: The Random Surfer Model

The damping factor models this "boredom"

2.1.2. Intuitive justification

PageRank can be thought of as a model of user behavior. We assume there is a “random surfer” who is given a Web page at random and keeps clicking on links, never hitting “back” but eventually gets bored and starts on another random page. The probability that the random surfer visits a page is its PageRank.

The anatomy of a large-scale hypertextual Web search engine (1998)

Damping Factor

$$0.9\begin{bmatrix} 0 & 1/6 & 1/3 & 0 & 0 & 0 \\ 1/2 & 1/6 & 1/3 & 0 & 0 & 0 \\ 1/2 & 1/6 & 0 & 0 & 0 & 0 \\ 0 & 1/6 & 0 & 0 & 1/2 & 1 \\ 0 & 1/6 & 1/3 & 1/2 & 0 & 0 \\ 0 & 1/6 & 0 & 1/2 & 1/2 & 0 \end{bmatrix} + \frac{0.1}{6}\mathbf{1} = \begin{bmatrix} 1/60 & 1/6 & 19/60 & 1/60 & 1/60 & 1/60 \\ 7/15 & 1/6 & 19/60 & 1/60 & 1/60 & 1/60 \\ 7/15 & 1/6 & 1/60 & 1/60 & 1/60 & 1/60 \\ 1/60 & 1/6 & 1/60 & 1/60 & 7/15 & 11/12 \\ 1/60 & 1/6 & 19/60 & 7/15 & 1/60 & 1/60 \\ 1/60 & 1/6 & 1/60 & 7/15 & 7/15 & 1/60 \end{bmatrix}$$

If $\alpha = 0.1$, then every zero gets increased slightly so that there is always some change jumping to a random node.

This is a reasonable model,
but it's also strategic

Recall: Convergence

Recall: Convergence

Definition. For a Markov chain with stochastic matrix A, an initial state $\mathbf{v}_0$ **converges** to the state $\mathbf{v}$ if $\lim_{k \to \infty} A^k \mathbf{v}_0 = \mathbf{v}$.

Recall: Convergence

Definition. For a Markov chain with stochastic matrix A, an initial state $\mathbf{v}_0$ **converges** to the state $\mathbf{v}$ if $\lim_{k \to \infty} A^k \mathbf{v}_0 = \mathbf{v}$.

As we repeatedly multiply $\mathbf{v}_0$ by A, we get closer and closer to $\mathbf{v}$ (in the limit).

Recall: Regular Stochastic Matrices

Recall: Regular Stochastic Matrices

Definition. A stochastic matrix A is **regular** if A^k has all positive entries for *some nonnegative* k.

Recall: Regular Stochastic Matrices

Definition. A stochastic matrix A is **regular** if A^k has all positive entries for *some nonnegative* k.

Theorem. A regular stochastic matrix P has a unique steady state, and

<u>every</u> probability vector converges to it

Damping Factor and regularity

$$0.9\begin{bmatrix} 0 & 1/6 & 1/3 & 0 & 0 & 0 \\ 1/2 & 1/6 & 1/3 & 0 & 0 & 0 \\ 1/2 & 1/6 & 0 & 0 & 0 & 0 \\ 0 & 1/6 & 0 & 0 & 1/2 & 1 \\ 0 & 1/6 & 1/3 & 1/2 & 0 & 0 \\ 0 & 1/6 & 0 & 1/2 & 1/2 & 0 \end{bmatrix} + \frac{0.1}{6}\mathbf{1} = \begin{bmatrix} 1/60 & 1/6 & 19/60 & 1/60 & 1/60 & 1/60 \\ 7/15 & 1/6 & 19/60 & 1/60 & 1/60 & 1/60 \\ 7/15 & 1/6 & 1/60 & 1/60 & 1/60 & 1/60 \\ 1/60 & 1/6 & 1/60 & 1/60 & 7/15 & 11/12 \\ 1/60 & 1/6 & 19/60 & 7/15 & 1/60 & 1/60 \\ 1/60 & 1/6 & 1/60 & 7/15 & 7/15 & 1/60 \end{bmatrix}$$

After damping, the matrix is regular.

It has a *unique steady state.*

The Algorithm (High Level)

PageRank

1. Build a graph encoding the websites and their links for the query we're given.
2. Build the adjacency matrix for this graph.
3. Turn boundaries into reflectors.
4. Normalize the matrix.
5. *Add a damping factor.*
6. Compute the eigenvector for the largest eigenvalues.
7. Order indices according to the entries of this vector.

np.linalg.eig(a)

(more on this later)

demo

The Issue

This is way too slow in practice.

And we don't need every eigenvector.

Recall AGAIN: Regular Stochastic Matrices

Recall AGAIN: Regular Stochastic Matrices

Definition. A stochastic matrix A is **regular** if A^k has all positive entries for *some nonnegative* k.

Recall AGAIN: Regular Stochastic Matrices

Definition. A stochastic matrix A is **regular** if A^k has all positive entries for *some nonnegative* k.

Theorem. A regular stochastic matrix P has a unique steady state, and

<u>every</u> probability vector converges to it

The Power Method

The Easiest Idea with the Most Intense Name

The Easiest Idea with the Most Intense Name

By regularity, we know that $A^k\mathbf{v}$ converges to the **unique steady state** starting at **any vector.**

The Easiest Idea with the Most Intense Name

By regularity, we know that $A^k\mathbf{v}$ converges to the **unique steady state** starting at **any vector.**

So...let's do that.

The Easiest Idea with the Most Intense Name

By regularity, we know that $A^k\mathbf{v}$ converges to the **unique steady state** starting at **any vector.**

So...let's do that.

Let's multiply <u>any</u> vector a bunch of times by A.

The Easiest Idea with the Most Intense Name

By regularity, we know that $A^k\mathbf{v}$ converges to the **unique steady state** starting at **any vector**.

So...let's do that.

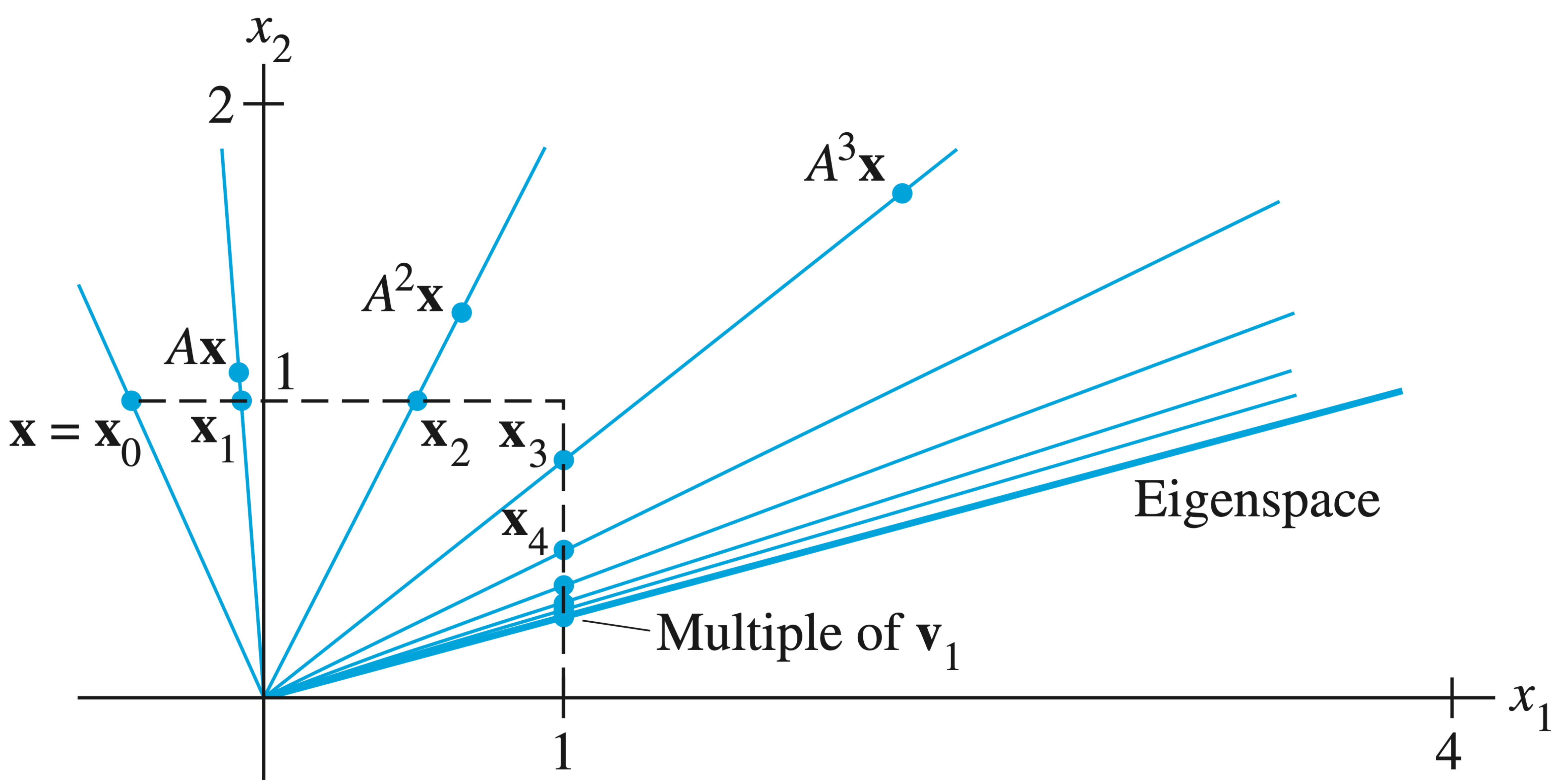
Let's multiply <u>any</u> vector a bunch of times by A.

Since $A^k\mathbf{v}$ approximates the steady-state, this will likely be a reasonably close solution.

Power Methods

Power methods are common in computational linear algebra because matrix multiplication is highly optimized.

They only give approximate solutions. But they can be very good, and they can be obtained very quickly.



Linear Algebra and its Applications, Lay, Lay, McDonald

The Power Method

```
FUNCTION steady_state_power_method(A):
    v ← random vector (or just 1)
    scale v so that it is a probability vector
    WHILE TRUE:
        v ← Av
```

The Power Method

```
FUNCTION steady_state_power_method(A):
    v ← random vector (or just 1)
    scale v so that it is a probability vector
    WHILE TRUE:
        v ← Av
```

When should we stop?

Termination Conditions

Option 1. *(Timeout)* Run for some fixed amount of time.

Option 2. *(Error tolerance)* Run until the change to the vector is very small.

The Power Method (Error Tolerance)

FUNCTION steady_state_power_method(A, ϵ):

    $\mathbf{v} \leftarrow$ random vector (or just $\mathbf{1}$)

    scale $\mathbf{v}$ so that it is a probability vector

    $\mathbf{v}' = Av$

    WHILE $\sum_{i=1}^{n} |\mathbf{v}_i - \mathbf{v}'_i| > \epsilon$: *# while the absolute difference between the last two approximations is large*

        $\mathbf{v}', \mathbf{v} \leftarrow A\mathbf{v}, \mathbf{v}'$

    RETURN $\mathbf{v}'$