Link Analysis

CS 4422/7263 Information Retrieval

Jiho Noh

Kennesaw State University

TOPICS

- Intro: Graphs
- Link Analysis
- PageRank

Slides are selectively adapted from the following lecture slides:

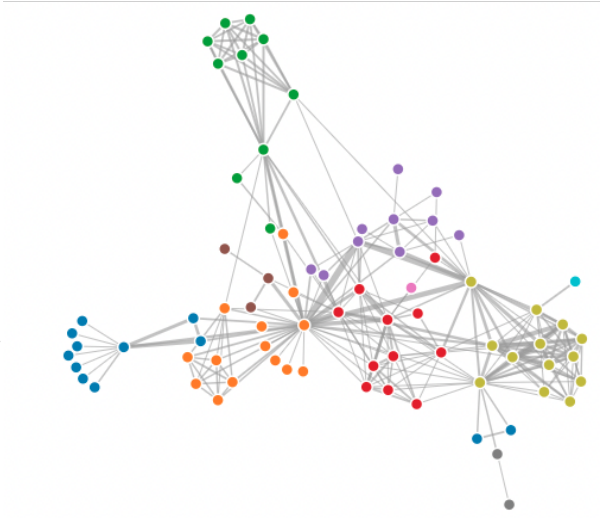
- *"Link Analysis: PageRank" by Jure Leskovec*

KENNESAW STATE UNIVERSITY

The Web as a Graph

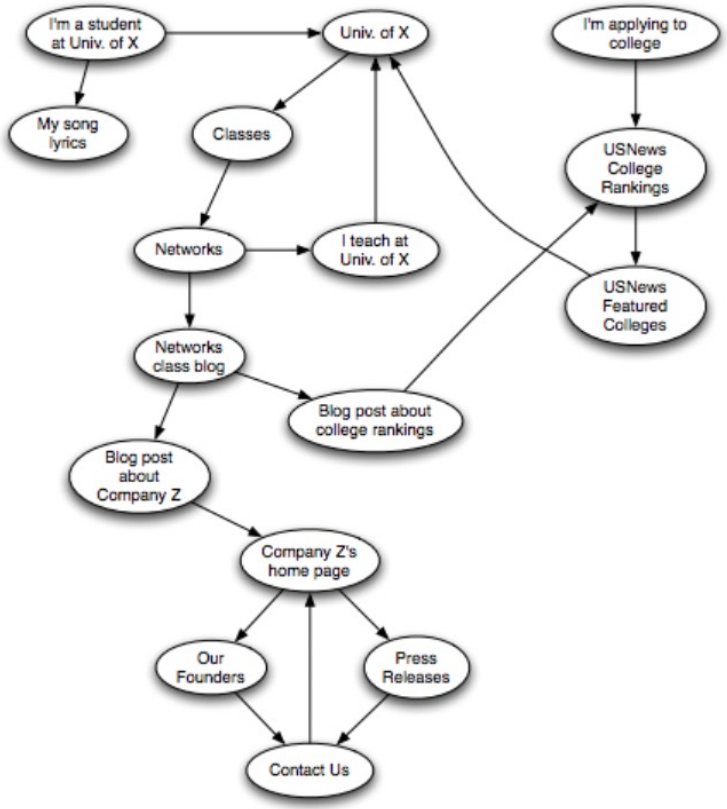
The Web as a graph

- *How does the Web graph look like?*
  - Web as a Graph:
    - Nodes: Web pages
    - Edges: Hyperlinks
- Side issues:
  - Dynamic pages generated on the fly
  - "Dark matter" — Inaccessible
  - Database generated pages



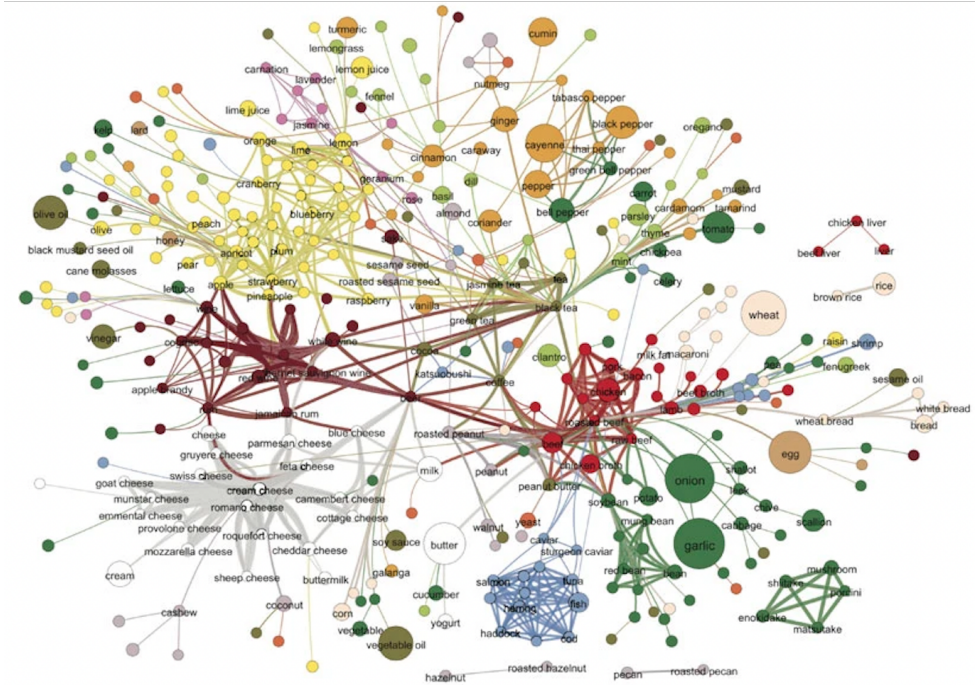
The Web as a graph

- In early days of the Web, links were **navigational**
- Today, many links are **transactional** (used not to navigate from page to page, but to post, comment, like, buy, etc.)



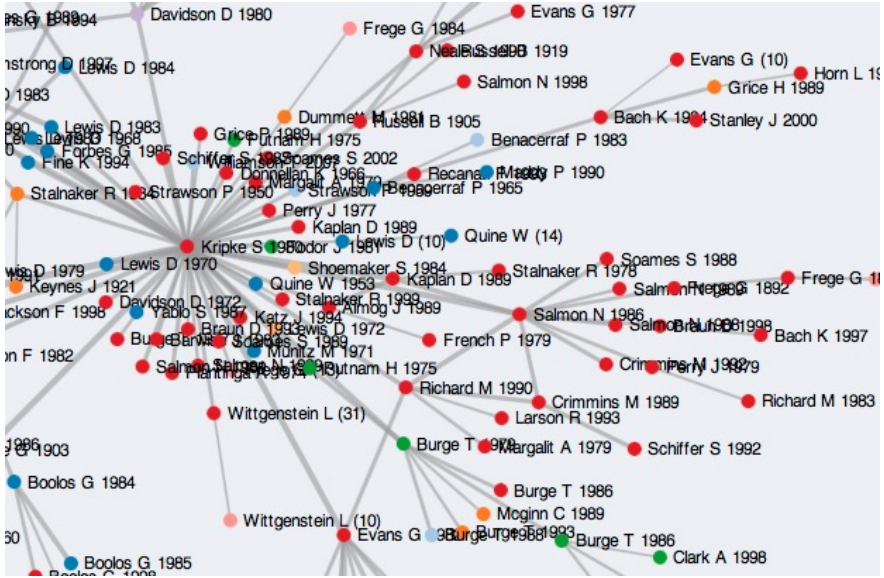
Other information networks — Flavor Network

- Node: flavor
- Edge: shared compounds
- Color: flavor category



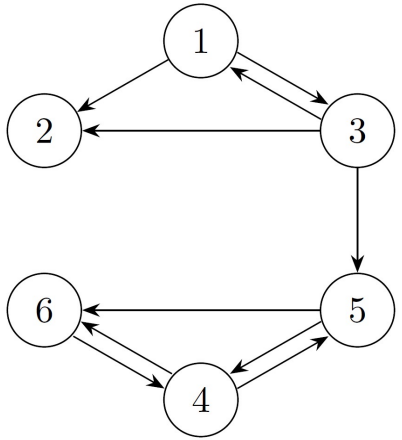
Other information networks — Citation Network

- Node: scholarly article
- Edge: citations
- Color: community



The Web graph — Conceptualization

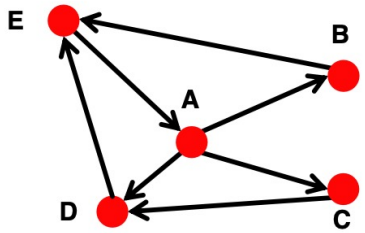
- Web as a Directed Graph [Broder et al., 2000]
- Given a node *v*,
  - What nodes can *v* reach?
  - What other nodes can reach *v*?

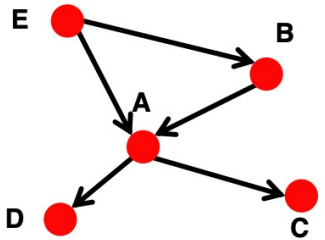


- IN(v) = { w | w can reach v}
  - e.g., IN(5) = {1, 3, 4, 5, 6}
- OUT(v) = { w | v can reach w}
  - e.g., OUT(5) = {4, 5, 6}

Types of directed graphs

- Strongly connected:
  - Any node can reach any node via a directed path
  - *IN(A) = OUT(A) = {A, B, C, D, E}*
- Directed Acyclic Graph (DAG):
  - Has no cycles: if *u* can reach *v*, then *v* cannot reach *u*





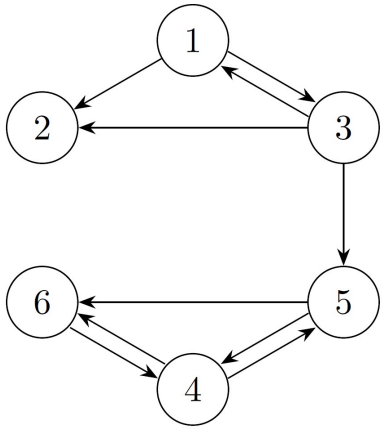
- Any directed graph (the Web) can be expressed in terms of these two types!

How is the Web constructed? A big strongly connected graph or a DAG?

Strongly Connected Components (SCC)

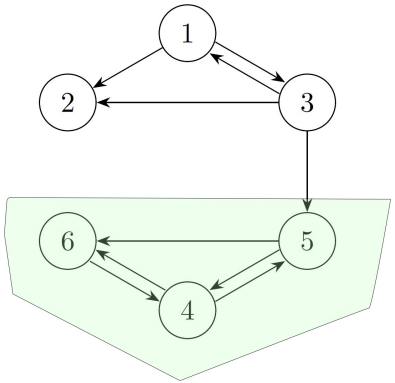
- **A Strongly Connected Component (SCC)** is a set of nodes *S* such that:
  - Every pair of nodes in a strongly connected component S can reach each other
  - There is no larger set containing *S* with this property

Find the SSC's in this graph. (note, a node can reach to itself)

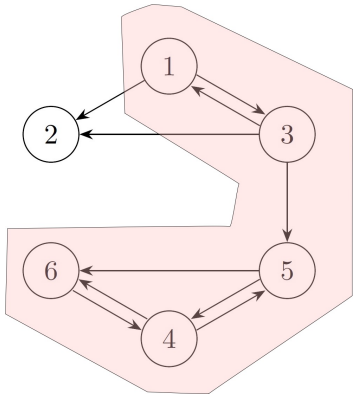


Finding SCCs from a directed graph

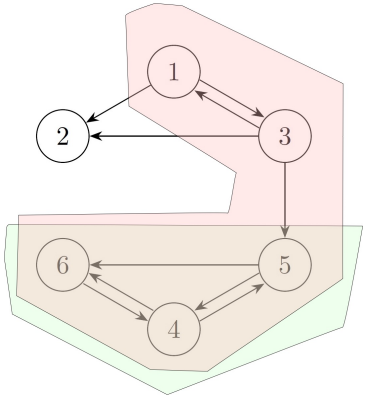
- Strongly connected: Any node can reach any node via a directed path



OUT (5) = {4, 5, 6}



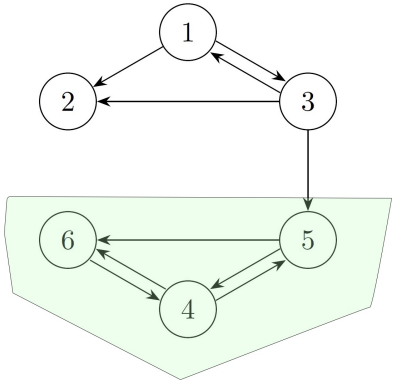
IN(5) = {1, 3, 4, 5, 6}

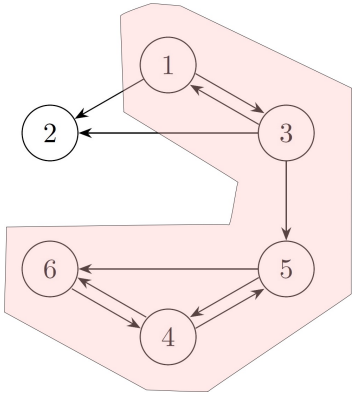


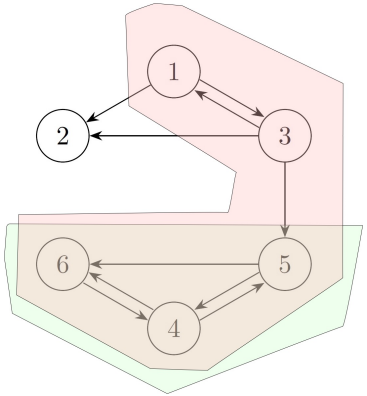
SCC(5) = OUT(5) ∩ IN(5) = {4, 5, 6}

Finding SCCs from a directed graph

- *How would you know the incoming links, how would you traverse them?*
- Observation:
  - Out(v) ∩ In(v) = Out(v, G) ∩ Out(v, G')
    - where G' is G with all edge directions flipped

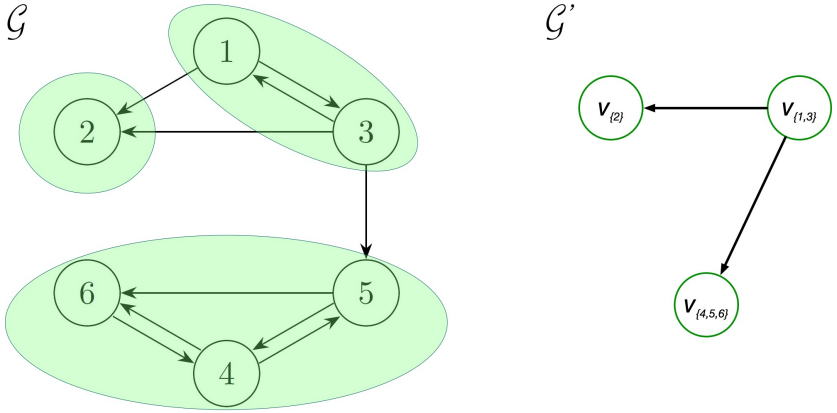






Directed graph properties

- Every directed graph is a DAG on its SCCs.
  1) SCCs partition the nodes of *G; i.e.*, each node is in exactly one SCC
  2) If we build a graph *G′* whose nodes are SCCs, and with an edge between nodes of *G′* if there is an edge between corresponding SCCs in *G*, then *G′* is a DAG



How do we find out SCCs computationally?

Graph structure in the Web

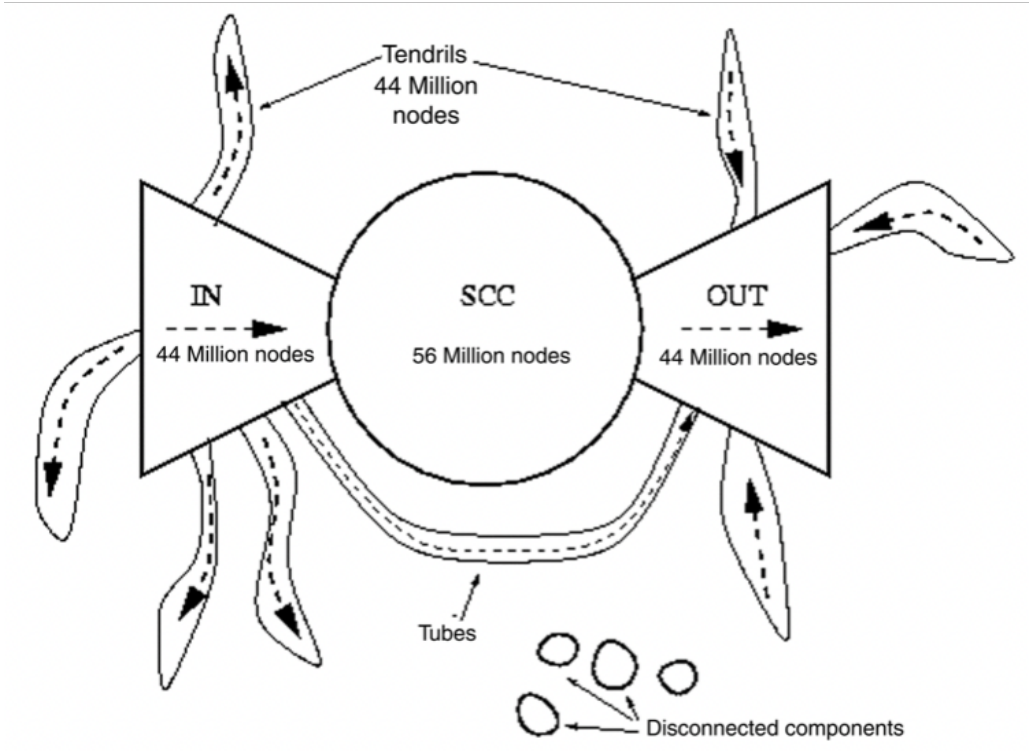
Broder et al.: **Altavista web crawl** (Oct '99)

- Analyzed 203 million nodes in May 1999 crawl
- BFS search over the web links
- Goal: *Take a large snapshot of the Web and try to understand how its SCCs "fit together" as a DAG*
- Results based on IN and OUT of a random node v:
  - OUT(v) ≈ 100 million (50% nodes)
  - IN(v) ≈ 100 million (50% nodes)
  - Largest SCC: 56 million (28% nodes)

Graph structure in the Web

- We can break the Web into four pieces:
  - The **largest SCC** where all the web pages can reach one another along directed link
  - **IN** which pages can reach the SCC
  - **OUT** which pages are accessible from the SCC
  - **TENDRILS** which pages cannot reach the SCC, and cannot be reached from the SCC

Bowtie structure of the Web



Link Analysis for IR

Link Analysis for IR

- All web pages are not equally "important"
- Links of the Web graph as the features in:
  - Scoring and ranking web pages
  - Clustering and identifying topical structures
  - Document classification — web pages that link to one another are likely to be on the same subject
  - Crawling policy — we can decide where to crawl next

PageRank — Links as Votes!

- PageRank, aka the Google Algorithm
- Invented by *Larry Page* and *Sergei Brin*, the founders of Google
- Intuitions:
  - A web page is more important if it has more links (*in-coming links? or out-going links?*)
  - We can rank web pages by their "importance" scores using the web graph link structure

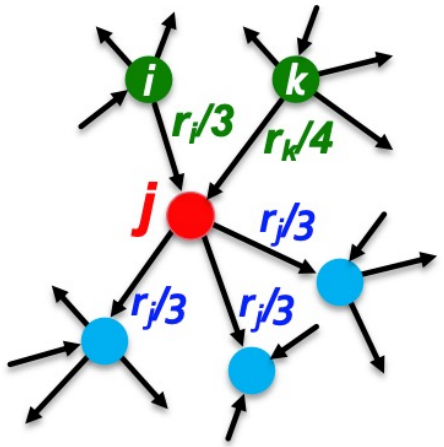
*How can we measure the **importance**?*

PageRank — Links as Votes!

- **Assumption 1**: Web page is more important if it has more in-links
  - https://www.whitehouse.gov/ (has 100,000+ in-links)
  - https://www.myhome.io/ (has 5 in-links)
- **Assumption 2**: Web page is more important if it has in-links from important pages
  - *Recursive problem!!*

PageRank — The "flow" model

- A "vote" from an important page is worth more:
  - Each link's vote is proportional to the importance of its source page
  - If page i with importance r_i has d_i out-links, each link gets r_i/d_i votes
  - Page j's own importance r_j is the sum of the votes on its in-links

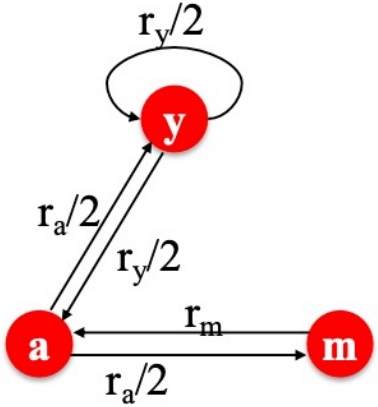


PageRank — The "flow" model

- Web page is more important if it has in-links from important pages
  - Define a "rank" score r_j for node j

$$r_j = \sum_{i \to j} \frac{r_i}{d_i}$$

- How to solve these equations?
  - You might wonder: Let's use Gaussian elimination to solve this system of linear equations. Bad idea! why?

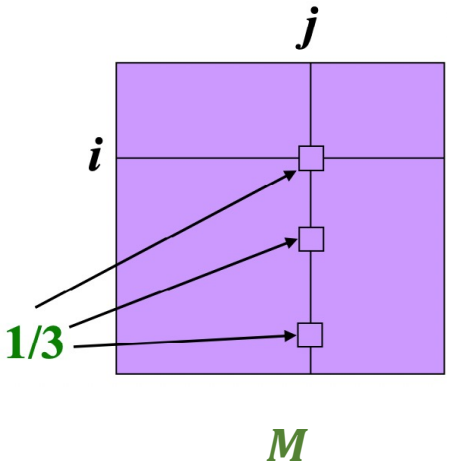


"Flow" equations:

$$r_y = r_y/2 + r_a/2$$
$$r_a = r_y/2 + r_m$$
$$r_m = r_a/2$$

PageRank — Matrix formulation

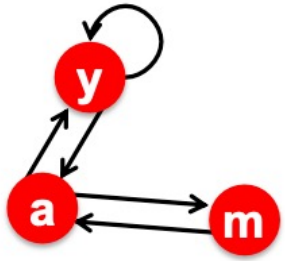
- **Stochastic adjacency matrix M**
  - Let page j have d_j out-links
  - If $j \to i$, then $\boldsymbol{M_{ij} = 1/d_j}$
  - "stochastic" means "having a random probability distribution"
  - M is a column stochastic matrix; that is columns sum to 1
- Rank vector r: An entry per page
  - r_i is the importance score of page i
  - $\sum_i r_i = 1$
- The flow equation can be written $\boldsymbol{r = M \cdot r}$



$$r_j = \sum_{i \to j} \frac{r_i}{d_i}$$

recursive problem

Example: PageRank flow equation



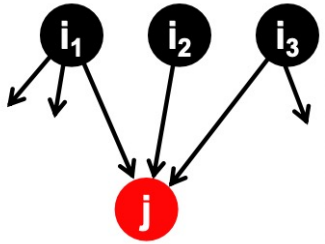
	r_y	r_a	r_m
r_y	½	½	0
r_a	½	0	1
r_m	0	½	0

$$r = M \cdot r$$

$$r_y = r_y/2 + r_a/2$$
$$r_a = r_y/2 + r_m$$
$$r_m = r_a/2$$

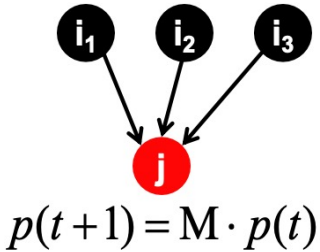
$$\begin{bmatrix} r_y \\ r_a \\ r_m \end{bmatrix} = \begin{bmatrix} ½ & ½ & 0 \\ ½ & 0 & 1 \\ 0 & ½ & 0 \end{bmatrix} \begin{bmatrix} r_y \\ r_a \\ r_m \end{bmatrix}$$

Random Walk interpretation



- *Imagine a random web surfer*
  - At any time $\boldsymbol{t}$, surfer is on some page i
  - At time $\boldsymbol{t} + \mathbf{1}$, the surfer follows an out-link from $\boldsymbol{i}$ uniformly at random
  - Ends up on some page $\boldsymbol{j}$ linked from $\boldsymbol{i}$
  - Process repeats indefinitely
- Let:
  - $\boldsymbol{p}(\boldsymbol{t})$ ... vector whose i^{th} coordinate is the prob. that the surfer is at page i at time t
  - So, $\boldsymbol{p}(\boldsymbol{t})$ is a probability distribution over pages

The Stationary Distribution



- **Where is the surfer at time *t+1*?**
  - Follow a link uniformly at random
    $$p(t+1) = M \cdot p(t)$$
  - Suppose the random walk reaches a state
    $$p(t+1) = M \cdot p(t) = p(t)$$
    then $p(t)$ is **stationary distribution** of a random walk
- **Our original rank vector $\boldsymbol{r}$** satisfies $\boldsymbol{r} = \boldsymbol{M} \cdot \boldsymbol{r}$
  - So, $\boldsymbol{r}$ is a stationary distribution for the random walk

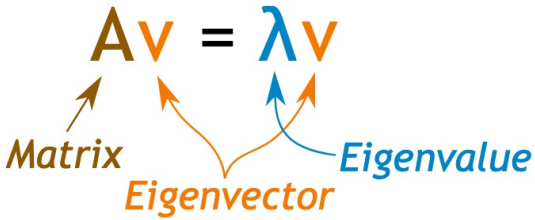
Stationary distribution

A stationary distribution is a specific entity which is unchanged by the effect of some matrix or operator: it need not be unique. Thus stationary distributions are related to eigenvectors for which the eigenvalue is unity.

PageRank: How to solve?

Eigenvector formulation

- The flow equation: $\boldsymbol{r = M \cdot r}$



*Use the **Power iteration method** to solve for $\boldsymbol{r}$*

- So, the rank vector $\boldsymbol{r}$ is an eigenvector of the stochastic web matrix $\boldsymbol{M}$
  - Starting from any vector $\boldsymbol{r^{(0)}}$, the limit $\boldsymbol{M(M(\ldots M(M\, r^{(0)})))}$ is the long-term distribution of the web surfers.
  - The limiting distribution is the dominant eigenvector of $\boldsymbol{M}$, which is the **PageRank**
- If $\boldsymbol{r}$ is the limit of $\boldsymbol{MM \ldots M\, r^{(0)}}$, then $\boldsymbol{r}$ is an eigenvector of $\boldsymbol{M}$ with eigenvalue 1

PageRank — Power Iteration Method

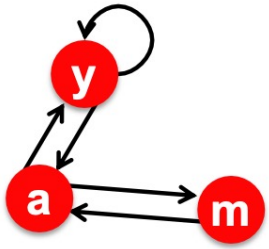
- Given a web graph with *n* nodes, where the nodes are pages and edges are hyperlinks
  1. Assign each node an initial page rank
  2. Calculate the page rank of each node

$$r_j^{(t+1)} = \sum_{i \to j} \frac{r_i^{(t)}}{d_i}$$

  3. Repeat until convergence ($\sum_i \left| r_i^{(t+1)} - r_i^{(t)} \right| < \varepsilon$)

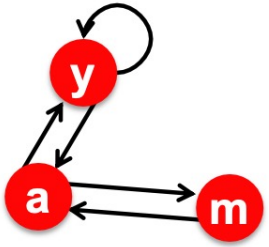
PageRank computation example

- Given a web graph with *n* nodes, where the nodes are pages and edges are hyperlinks
  1. Assign each node an initial page rank
     $$r_j \leftarrow 1/N$$
  2. Calculate the page rank of each node
     $$r_j' \leftarrow \sum_{i \to j} \frac{r_j}{d_i}$$
  3. Repeat until convergence ($\sum_i \left| r_i^{(t+1)} - r_i^{(t)} \right| < \varepsilon$) :
     $$if\ |r\ - r'| > \varepsilon, then\ r \leftarrow r'\ and\ go\ to\ step\ 1$$



	y	a	m
y	½	½	0
a	½	0	1
m	0	½	0

PageRank computation example



	y	a	m
y	½	½	0
a	½	0	1
m	0	½	0

$$\begin{bmatrix} r_y \\ r_a \\ r_m \end{bmatrix} = \begin{matrix} 1/3 & 1/3 & 5/12 & 9/24 & & 6/15 \\ 1/3 & 3/6 & 1/3 & 11/24 & \ldots & 6/15 \\ 1/3 & 1/6 & 3/12 & 1/6 & & 3/15 \end{matrix}$$

Iteration 0, 1, 2, …

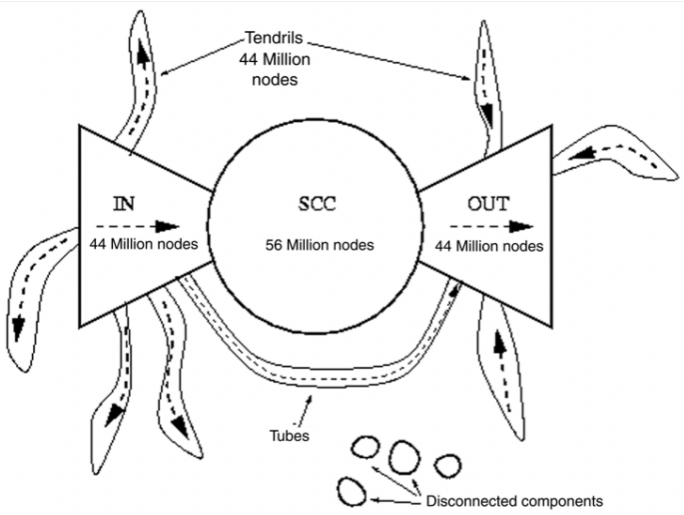
PageRank: Three questions

$$r_j^{(t+1)} = \sum_{i \to j} \frac{r_i^{(t)}}{d_i}$$ or equivalently $$r = M \cdot r$$

- Does this converge?
- Does it converge to what we want?
- Are results reasonable?

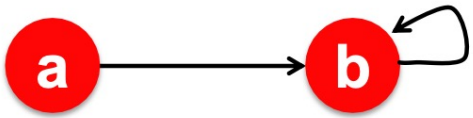
Two problems:

1. Some pages are **dead ends** (have no out-links)
   - Such pages cause importance to "leak out"
2. **Spider traps**
   - All out-links are within the group
   - Eventually spider traps absorb all importance



Does this converge?

- **The "Spider trap" problem**
  - Spier trap will absorb all the importance



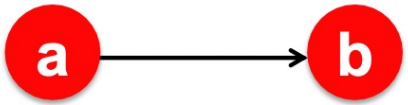
$$r_j^{(t+1)} = \sum_{i \to j} \frac{r_i^{(t)}}{\mathrm{d_i}}$$

- **Example:**

Iteration:	0,	1,	2,	3…
r_a	1	0	0	0
r_b	0	1	1	1

Does it converge to what we want?

- **The "Dead end" problem:**
  - Importance is leaking from the dead end



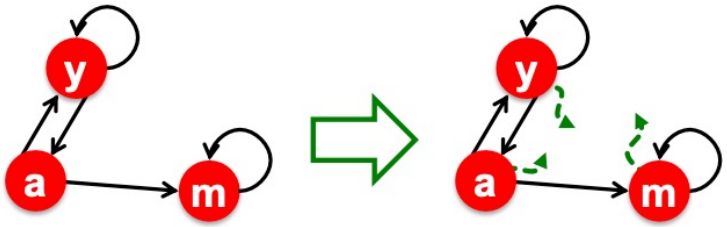
$$r_j^{(t+1)} = \sum_{i \to j} \frac{r_i^{(t)}}{\mathrm{d_i}}$$

- **Example:**

Iteration:	0,	1,	2,	3…
r_a	1	0	0	0
r_b	0	1	0	0

Solution to Spider Traps

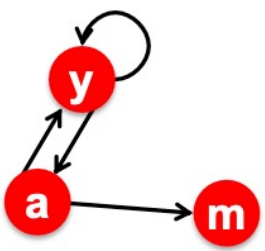
- The Google solution for spider traps:
  - At each time step, the random surfer has two options
    - With prob. β, follow a link at random
    - With prob. $1 - \beta$, jump to a random page
    - Common values for β are in the range 0.8 to 0.9
  - Surfer will teleport out of spider trap within a few time steps

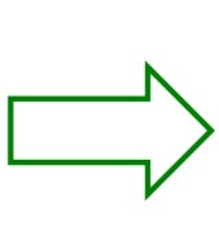


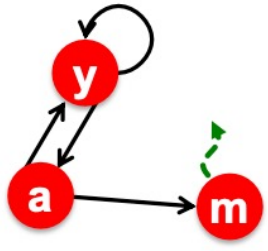
Solution to Dead Ends

Teleporting

- Follow random teleport links with total probability 1.0 from dead-ends
  - Adjust matrix accordingly







	y	a	m
y	½	½	0
a	½	0	0
m	0	½	0

	y	a	m
y	½	½	⅓
a	½	0	⅓
m	0	½	⅓

Why teleports solve the problem?

Why are dead-ends and spider traps a problem and why do teleports solve the problem?

- **Spider-traps** are not a problem, but with traps PageRank scores are not what we want
  - Solution: Never get stuck in a spider trap by teleporting out of it in a finite number of steps
- **Dead-ends** are a problem
  - The matrix is not column stochastic so our initial assumptions are not met
  - Solution: Make matrix column stochastic by always teleporting when there is nowhere else to go

Solution: Random Teleports

Results of Teleporting

- Now cannot get stuck locally.
- There is a long-term rate at which any page is visited
- *How do we compute this visit rate?*
  - Google's solution: **PageRank equation** [Brin-Page, '98]

$$r_j = \sum_{i \to j} \beta \frac{r_i}{d_i} + (1 - \beta) \frac{1}{N}$$

  - Equivalently, the Google Matrix A:

$$A = \beta M + (1 - \beta) \left[\frac{1}{N}\right]_{N \times N}$$

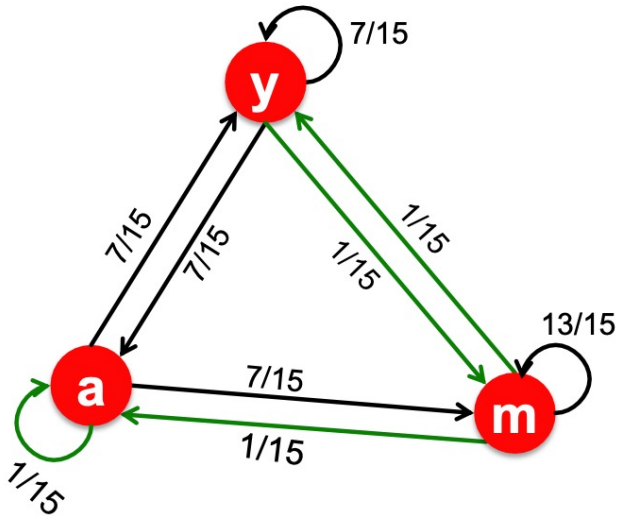
Solution: Random Teleports

- Equivalently, the Google Matrix A:

$$A = \beta M + (1 - \beta)\left[\frac{1}{N}\right]_{N \times N}$$

  - We have a recursive problem: $\boldsymbol{r = A \cdot r}$
    - And the Power iteration method still works!
  - In practice, $\beta = 0.8, 0.9$

Random Teleports example ($\beta = 0.8$)



$$0.8 \begin{bmatrix} 1/2 & 1/2 & 0 \\ 1/2 & 0 & 0 \\ 0 & 1/2 & 1 \end{bmatrix} + 0.2 \begin{bmatrix} 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \end{bmatrix}$$

M $\qquad$ $[1/N]_{NxN}$

$$\begin{array}{c} \\ y \\ a \\ m \end{array} \begin{bmatrix} 7/15 & 7/15 & 1/15 \\ 7/15 & 1/15 & 1/15 \\ 1/15 & 7/15 & 13/15 \end{bmatrix}$$

A

$$\begin{array}{c} y \\ a \\ m \end{array} = \begin{array}{ccccccc} 1/3 & 0.33 & 0.24 & 0.26 & & & 7/33 \\ 1/3 & 0.20 & 0.20 & 0.18 & \ldots & & 5/33 \\ 1/3 & 0.46 & 0.52 & 0.56 & & & 21/33 \end{array}$$

Computing PageRank

- The Google Matrix A is a dense matrix:
  - $r^{new} = A \cdot r^{old}$
  - Easy if we have enough main memory to hold A, r^{old}, r^{new}
  - Say N = 1 billion pages,
  - The computation problem is not trivial anymore
- We need to utilize sparse matrix computation

$$\mathbf{A} = \beta \cdot \mathbf{M} + (1-\beta)\,[1/N]_{NxN}$$

$$A = 0.8 \begin{bmatrix} ½ & ½ & 0 \\ ½ & 0 & 0 \\ 0 & ½ & 1 \end{bmatrix} + 0.2 \begin{bmatrix} 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \end{bmatrix}$$

$$= \begin{bmatrix} 7/15 & 7/15 & 1/15 \\ 7/15 & 1/15 & 1/15 \\ 1/15 & 7/15 & 13/15 \end{bmatrix}$$

Sparse matrix formulation

- We can rearrange the PageRank equation

$$A = \beta M + (1-\beta)\left[\frac{1}{N}\right]_{N\times N} \quad \rightarrow \quad r = \beta M \cdot r + \left[\frac{1-\beta}{N}\right]_N$$

- $\boldsymbol{M}$ **is a sparse matrix!** (with no dead-ends)
  - 10 links per node, approximately 10N entries
- So, in each iteration, we need to
  - compute $r^{new} = \beta M \cdot r^{old}$
  - then add a constant value (1-β)/N to each entry in r^{new}
  - Note, if M contains dead-ends then $\sum_j r_j^{new} < 1$ and we also have to renormalize r^{new} so that it sums to 1

PageRank: The complete algorithm

- **Input**: Given a graph $\mathcal{G}$ and parameter β
- **Output**: PageRank vector r^{new}

  - Set $r_j^{old} = 1/N$
  - repeat computation until convergence: $\sum_j \left| r_j^{new} - r_j^{old} \right| < \varepsilon)$
    - $\forall j$: $r'^{new}_j = \sum_{i \to j} \beta \frac{r'^{old}_i}{d_j}$
      - d_j is the out-degree of node j
    - Now, re-insert the leaked PageRank:
    - $\forall j$: $r_j^{new} = r'^{new}_j + \frac{1-S}{N}$ where: $S = \sum_j r'^{new}_j$
    - $r^{old} = r^{new}$

Example