CS 5683: Big Data Analytics

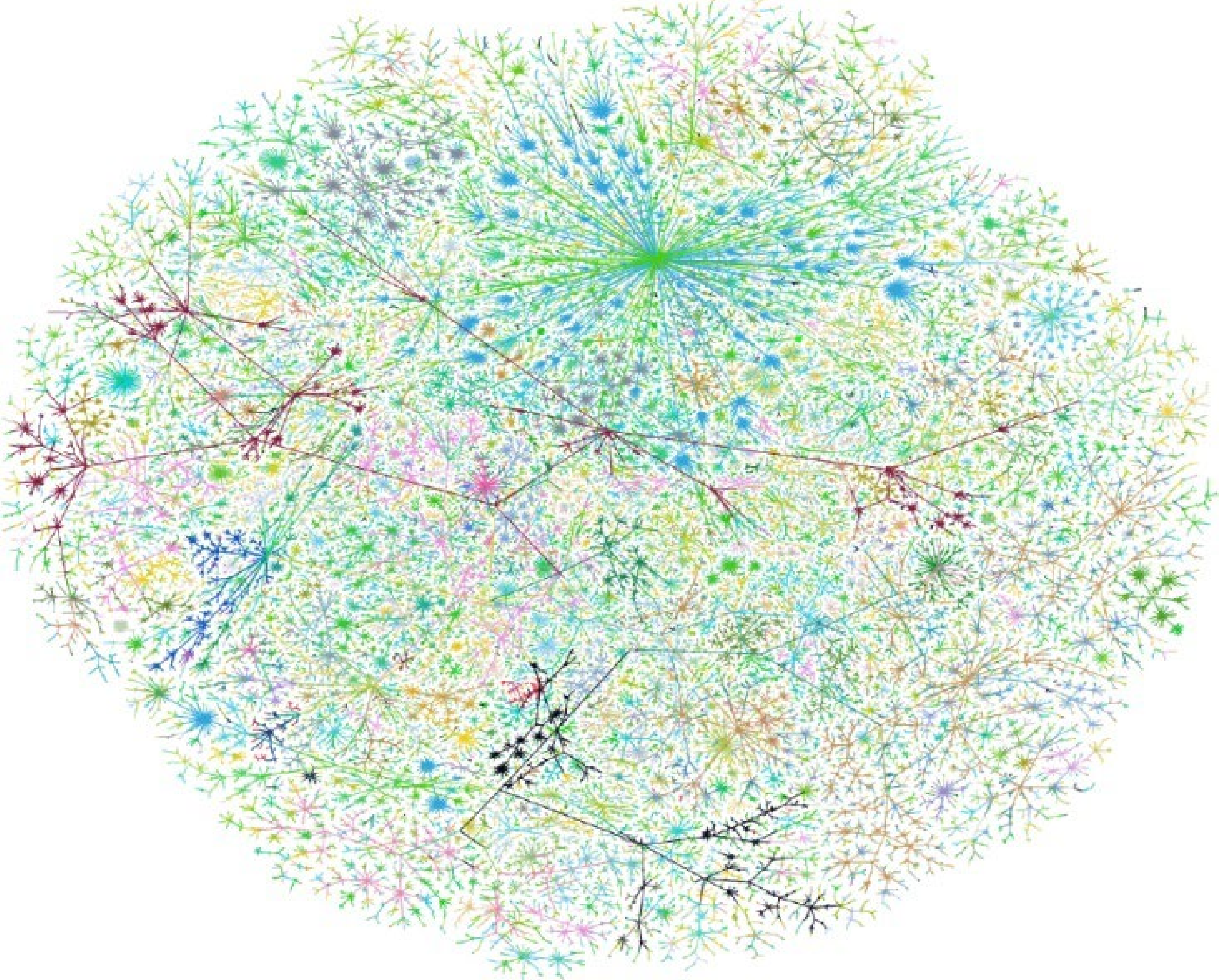
Analysis of Large Graphs: PageRank

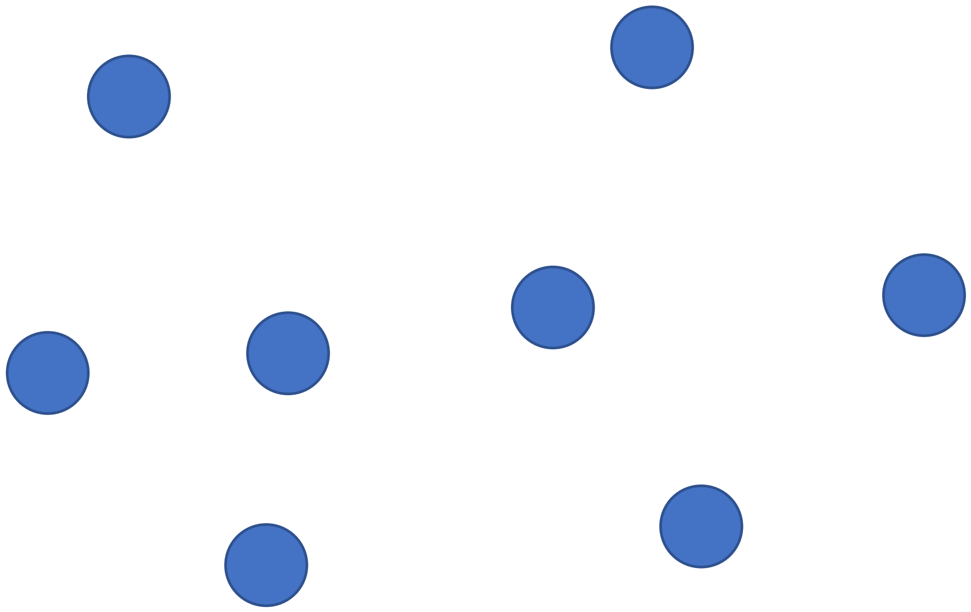
Arunkumar Bagavathi
Department of Computer Science
Oklahoma State university

Topics Overview

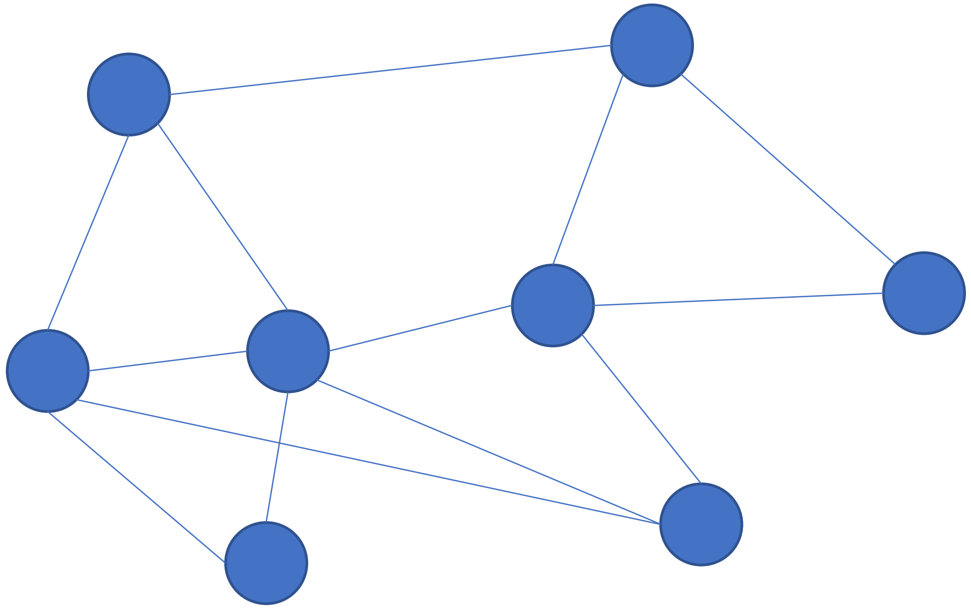
High. Dim. Data	Text Data	Graph Data	Others
Data Features	Clustering	PageRank	Data Streams Mining
Dimensionality Reduction	Non-linear Dim. Reduction	ML for Graphs	Intro. to Apache Spark
Application Rec. Systems	*Application* IR	Community Detection	

The Network!



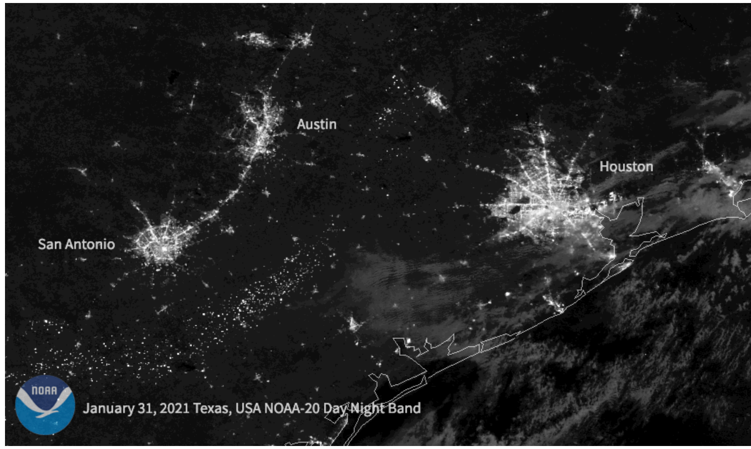


Entities/Nodes/Vertices



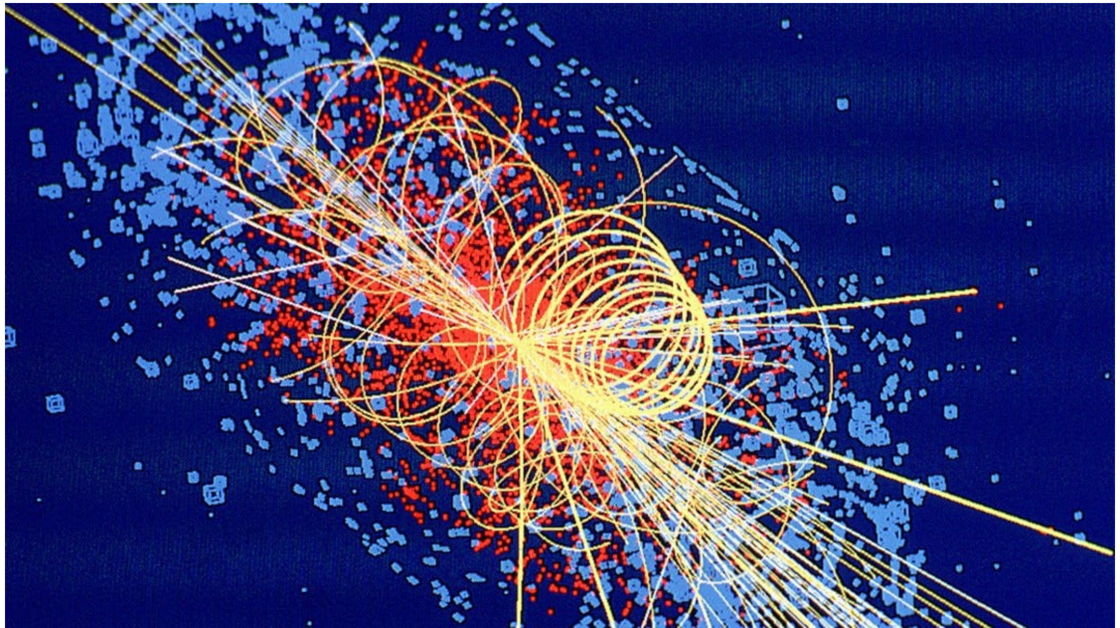
Network

Networks & Complex Systems



Complex systems are hopelessly around us:

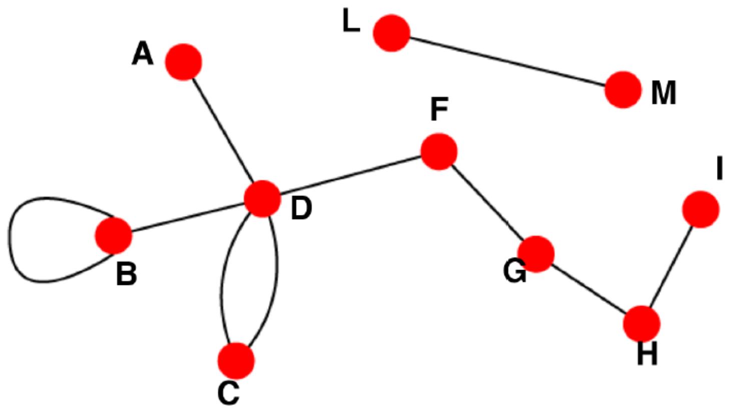
- **Society** is a collection of 7+ billion individuals
- **Communication systems** link electronic devices
- **Information** and **knowledge** is organized and linked
- Thousands of **genes** in our cells work together in a seamless fashion
- Our **thoughts** are hidden in the connections between billions of neurons in our brain



Types of Network Representations Directed vs. Undirected

Undirected

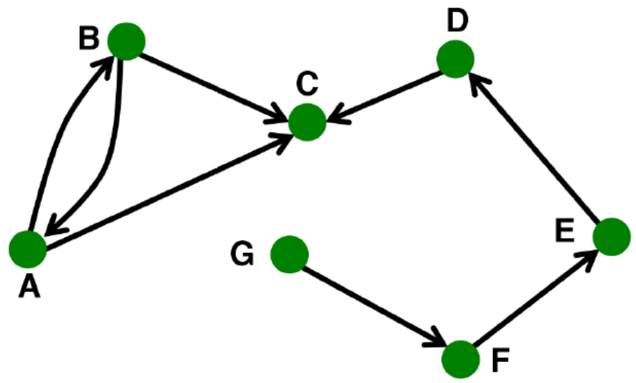
- **Links:** undirected (symmetrical, reciprocal)



- **Examples:**
  - Collaborations
  - Friendship on Facebook

Directed

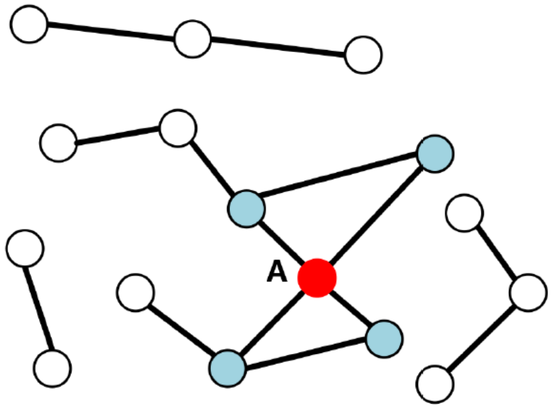
- **Links:** directed (arcs)



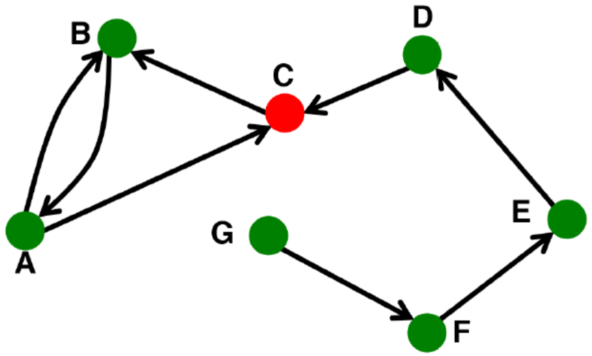
- **Examples:**
  - Phone calls
  - Following on Twitter

Node Degrees

Undirected



Directed



Source: Node with $k^{in} = 0$

Sink: Node with $k^{out} = 0$

Node degree, k_i: the number of edges adjacent to node i

$$k_A = 4$$

Avg. degree: $\bar{k} = \langle k \rangle = \frac{1}{N}\sum_{i=1}^{N} k_i = \frac{2E}{N}$

In directed networks we define an **in-degree** and **out-degree.**

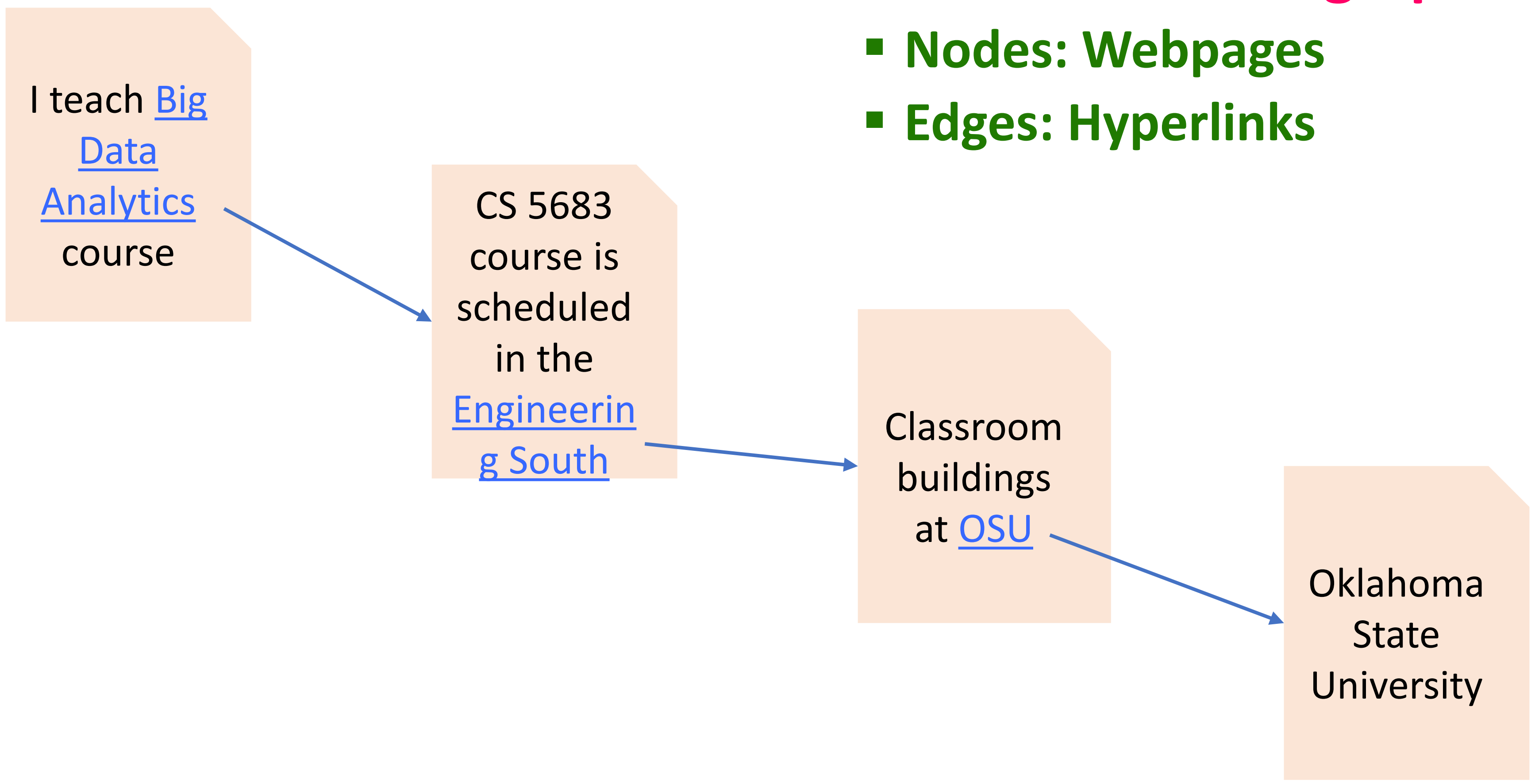
The (total) degree of a node is the sum of in- and out-degrees.

$$k_C^{in} = 2 \qquad k_C^{out} = 1 \qquad k_C = 3$$

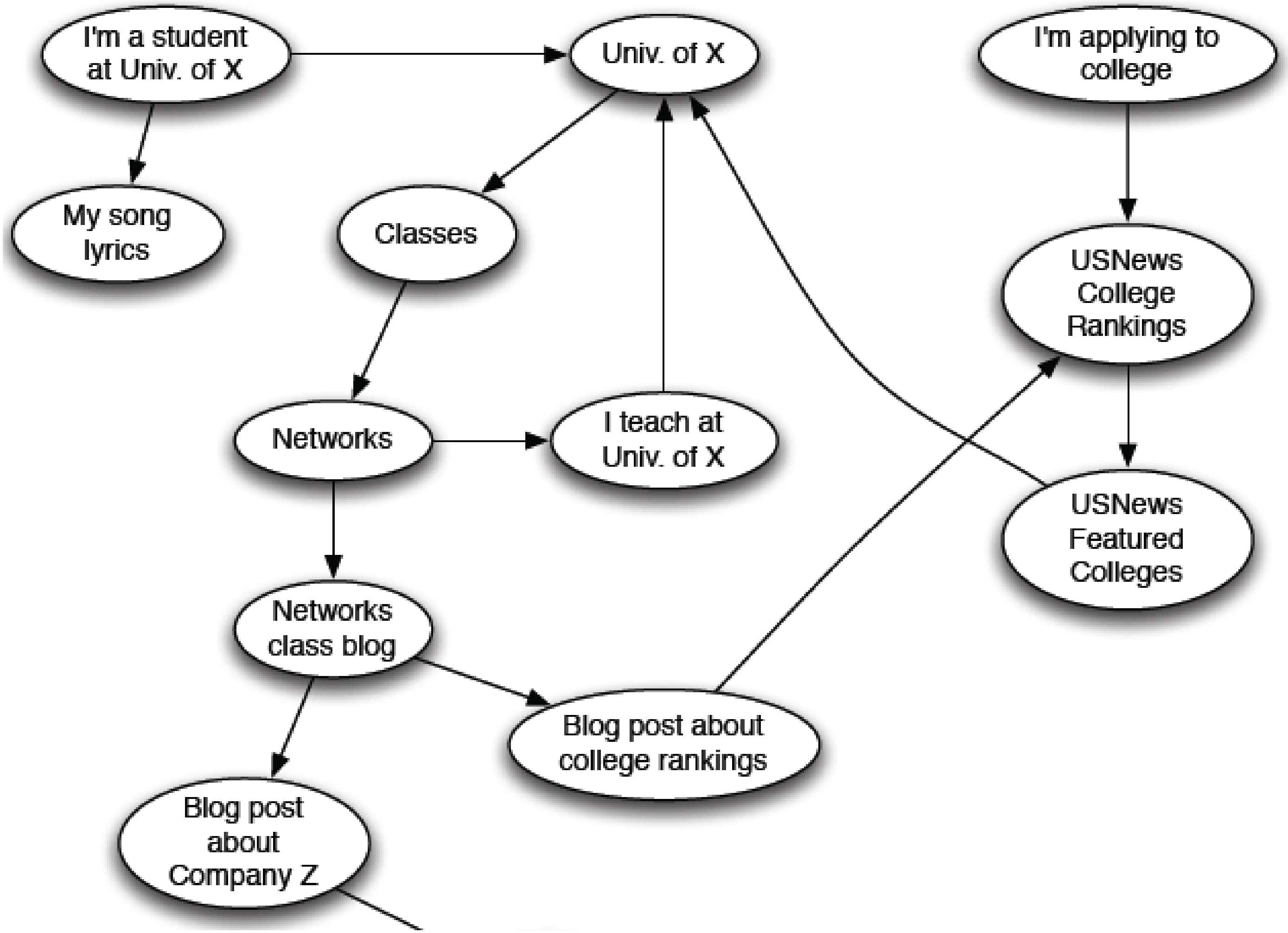
$$\bar{k} = \frac{E}{N} \qquad\qquad \overline{k^{in}} = \overline{k^{out}}$$

Web as a Graph

- **Web as a directed graph:**
  - **Nodes: Webpages**
  - **Edges: Hyperlinks**



Web as a Directed Graph



Broad Question

- **How to organize the Web?**

- **First try:** Human curated **Web directories**
  - Yahoo, DMOZ, LookSmart

- **Second try: Web Search**
  - **Information Retrieval** investigates:
    Find relevant docs in a small and trusted set
    - Newspaper articles, Patents, etc.
  - But: Web is **huge**, full of untrusted documents, random things, web spam, etc.

Web Search: 2 Challenges

2 challenges of web search:

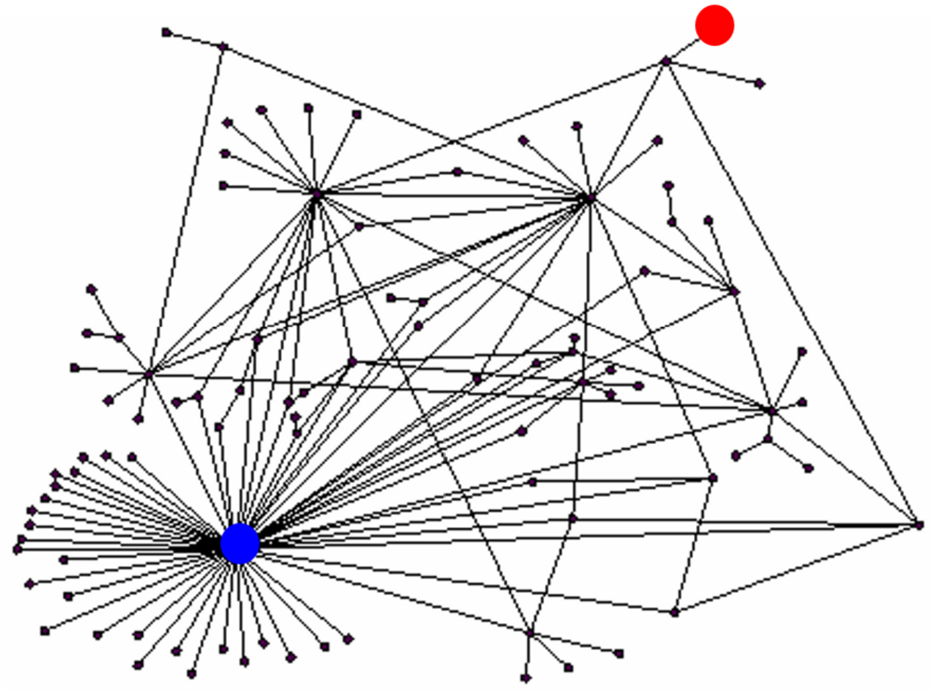
- **(1) Web contains many sources of information**
  Who to “trust”?
  - **Trick:** Trustworthy pages may point to each other!
- **(2) What is the “best” answer to query “newspaper”?**
  - No single right answer
  - **Trick:** Pages that actually know about newspapers might all be pointing to many newspapers

Ranking Nodes on the Graph

- **All web pages are not equally “important”**

- There is large diversity in the web-graph node connectivity.

- **Let’s rank the pages by the link structure!**



Link Analysis Algorithms

- We will cover the following **Link Analysis approaches** for computing **importance** of nodes in a graph:
  - Page Rank
  - Hubs and Authorities (HITS)
  - Topic-Specific (Personalized) Page Rank
  - Web Spam Detection Algorithms

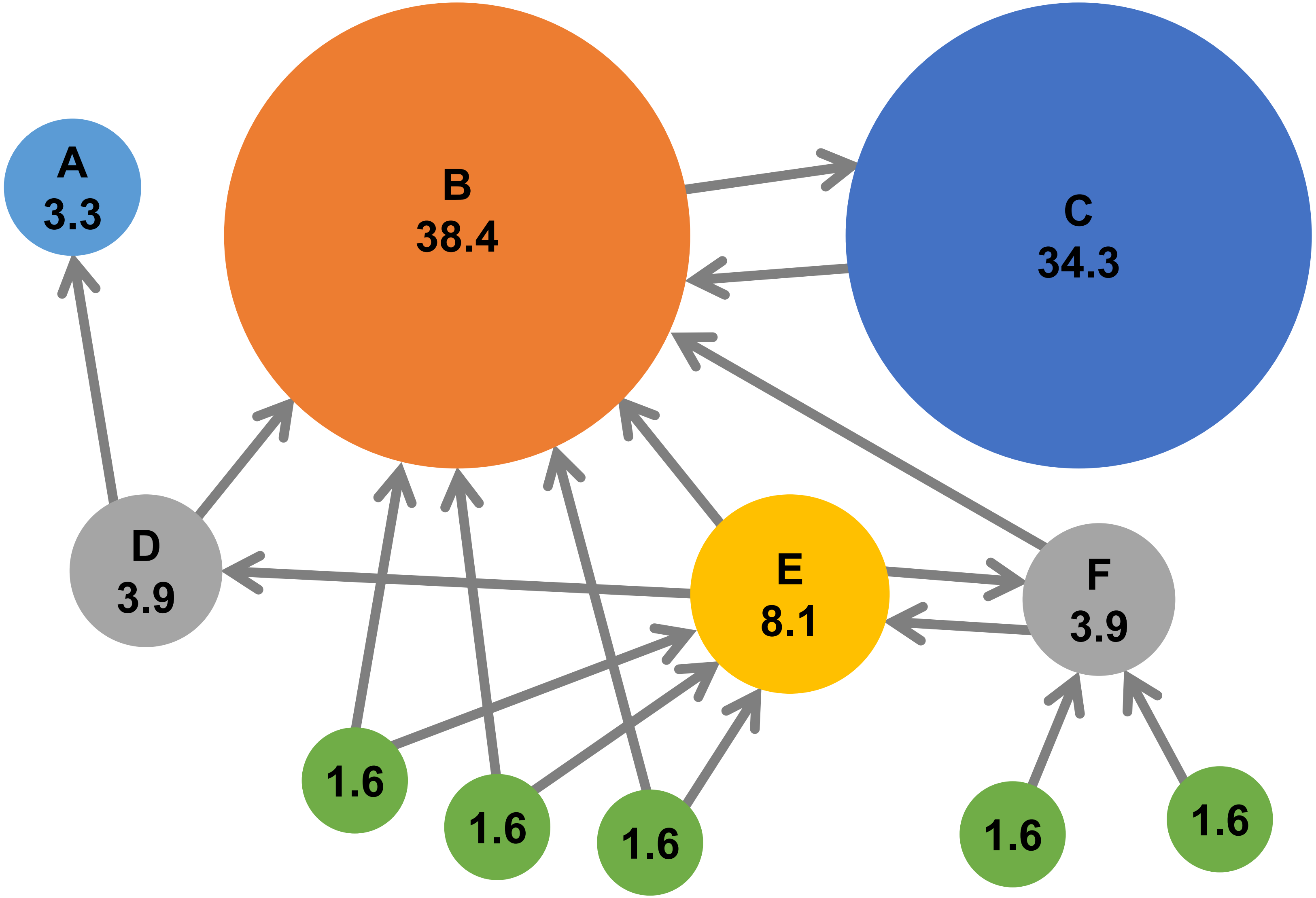
PageRank – Links as Votes

- **Idea: Links as votes**
  - **Page is more important if it has more links**
    - In-coming links? Out-going links?

- **Think of in-links as votes:**
  - www.okstate.edu has 1,000s of in-links
  - www.joe-schmoe.com has 1 in-link

- **Are all in-links are equal?**
  - **Links from important pages count more**
  - Recursive question!

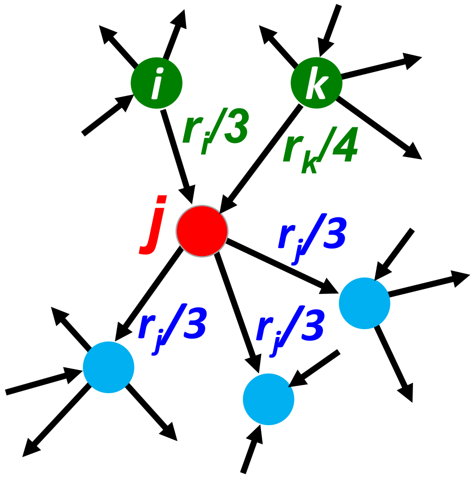
Example: PageRank Scores



Simple Recursive Formulation

- Each link's vote is proportional to the **importance** of its source page
- If page ***j*** with importance r_j has ***n*** out-links, each link gets r_j / n votes
- Page ***j***'s own importance is the sum of the votes on its in-links

$$r_j = r_i/3 + r_k/4$$

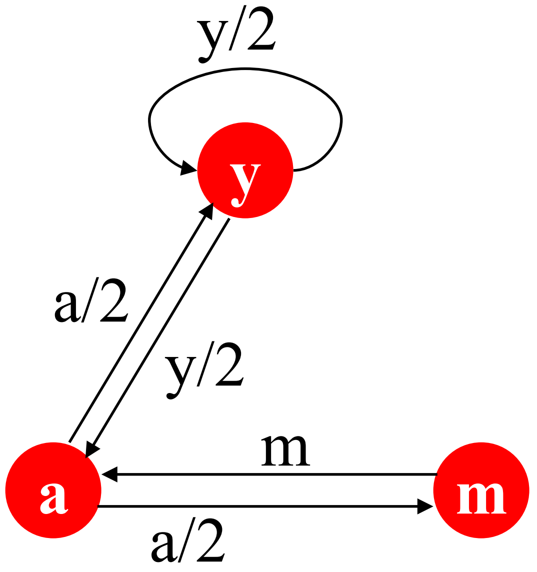


PageRank: The "Flow" Model

- **A "vote" from an important page is worth more**
- **A page is important if it is pointed to by other important pages**
- **Define a "rank" r_j for page j**

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

d_i … out-degree of node i



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

Solving the Flow Equations

- **3 equations, 3 unknowns, no constants**
  - No unique solution
  - All solutions equivalent modulo the scale factor
- **Additional constraint forces uniqueness:**
  - $\boldsymbol{r_y + r_a + r_m = 1}$
  - **Solution:** $\boldsymbol{r_y = \frac{2}{5},\ r_a = \frac{2}{5},\ r_m = \frac{1}{5}}$
- **Gaussian elimination method works for small examples, but we need a better method for large web-size graphs**

- **We need a new formulation!**

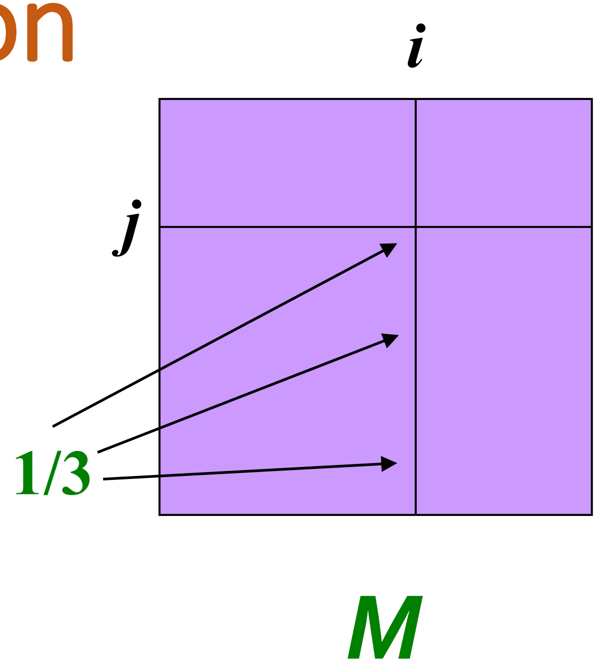
Flow equations:

$$\mathbf{r_y = r_y/2 + r_a/2}$$
$$\mathbf{r_a = r_y/2 + r_m}$$
$$\mathbf{r_m = r_a/2}$$

PageRank: Matrix Formulation

- **Stochastic adjacency matrix $\boldsymbol{M}$**
  - Let page i has d_i out-links
  - If $i \rightarrow j$, then $M_{ji} = \frac{1}{d_i}$ else $M_{ji} = 0$
    - $\boldsymbol{M}$ is a **column stochastic matrix**
      - Columns sum to 1
- **Rank vector $\boldsymbol{r}$:** vector with an entry per page
  - r_i is the importance score of page i
  - $\sum_i r_i = 1$
- **The flow equations can be written**

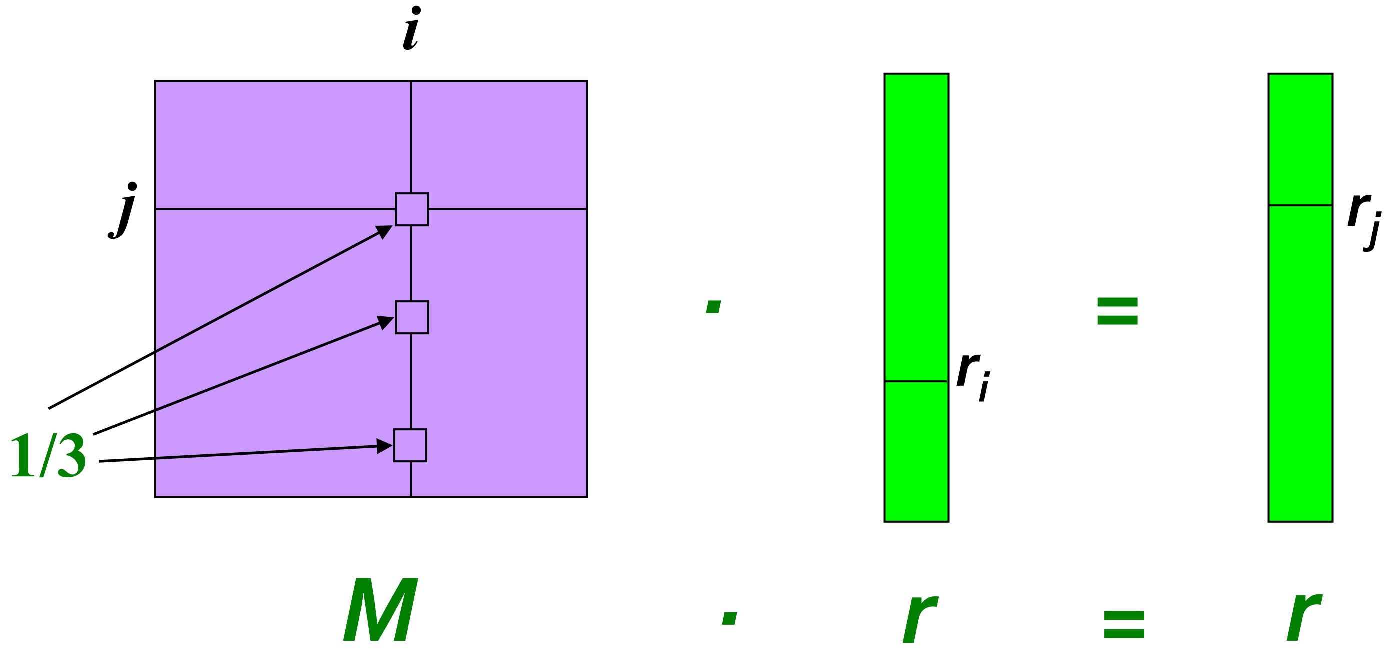
$$\boldsymbol{r} = \boldsymbol{M} \cdot \boldsymbol{r}$$



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

Example

- **Remember the flow equation:** $r_j = \sum_{i \to j} \frac{r_i}{\mathrm{d_i}}$
- **Flow equation in the matrix form**
  - $M \cdot r = r$
    - **Suppose page *i* links to 3 pages, including *j***



Eigenvector Formulation

- **The flow equations can be written**

$$\boldsymbol{r} = \boldsymbol{M} \cdot \boldsymbol{r}$$

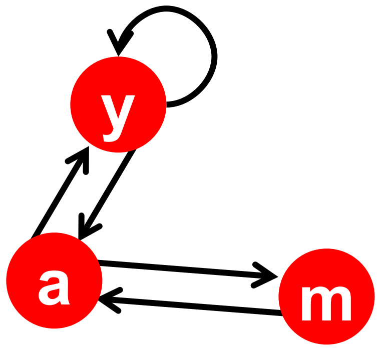
NOTE: $\boldsymbol{x}$ is an eigenvector with the corresponding eigenvalue $\boldsymbol{\lambda}$ if:

$$\boldsymbol{Ax} = \boldsymbol{\lambda x}$$

- So, the **rank vector** ***r*** is an **eigenvector** of the stochastic web matrix ***M***
  - In fact, its first or principal eigenvector, with corresponding eigenvalue ***1***
    - Largest eigenvalue of ***M*** is **1** since ***M*** is column stochastic (with non-negative entries)
      - *We know* ***r*** *is unit length and each column of* **M** *sums to one, so* $\boldsymbol{Mr} \le \mathbf{1}$

- **We can now efficiently solve for *r*!**
  The method is called Power iteration

Example: Flow Equations & M



	y	a	m
y	½	½	0
a	½	0	1
m	0	½	0

r* = *M*·*r

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

$$\begin{bmatrix} y \\ a \\ m \end{bmatrix} = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} & 0 \\ \frac{1}{2} & 0 & 1 \\ 0 & \frac{1}{2} & 0 \end{bmatrix} \begin{bmatrix} y \\ a \\ m \end{bmatrix}$$

Power Iteration Method

- **Given a web graph with *n* nodes, where the nodes are pages and edges are hyperlinks**

- **Power iteration:** a simple iterative scheme
  - Suppose there are *N* web pages
  - Initialize: $\mathbf{r}^{(0)} = [1/N,....,1/N]^T$
  - Iterate: $\mathbf{r}^{(t+1)} = \mathbf{M} \cdot \mathbf{r}^{(t)}$
  - Stop when $|\mathbf{r}^{(t+1)} - \mathbf{r}^{(t)}|_1 < \varepsilon$

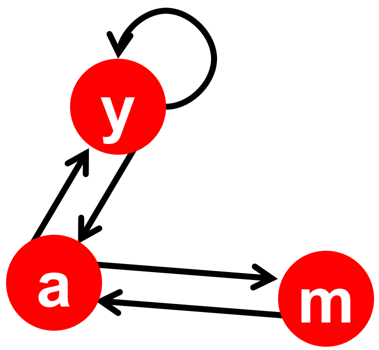
$$r_j^{(t+1)} = \sum_{i \to j} \frac{r_i^{(t)}}{\mathrm{d}_i}$$

d_i …. out-degree of node i

$|\mathbf{x}|_1 = \sum_{1 \le i \le N} |x_i|$ is the $\mathbf{L_1}$ norm
Can use any other vector norm, e.g., Euclidean

PageRank: How to solve?

- **Power Iteration:**
  - Set $r_j = 1/N$
  - **1:** $r'_j = \sum_{i \to j} \frac{r_i}{d_i}$
  - **2:** $r = r'$
  - Goto **1**



	y	a	m
y	½	½	0
a	½	0	1
m	0	½	0

$\mathbf{r_y = r_y/2 + r_a/2}$

$\mathbf{r_a = r_y/2 + r_m}$

$\mathbf{r_m = r_a/2}$

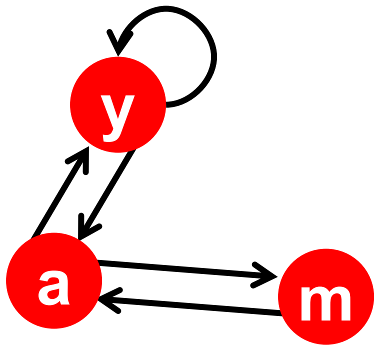
- **Example:**

$$\begin{pmatrix} r_y \\ r_a \\ r_m \end{pmatrix} = \begin{matrix} 1/3 \\ 1/3 \\ 1/3 \end{matrix}$$

Iteration 0, 1, 2, …

PageRank: How to solve?

- **Power Iteration:**
  - Set $r_j = 1/N$
  - **1:** $r'_j = \sum_{i\to j} \frac{r_i}{d_i}$
  - **2:** $r = r'$
  - Goto **1**



	y	a	m
y	½	½	0
a	½	0	1
m	0	½	0

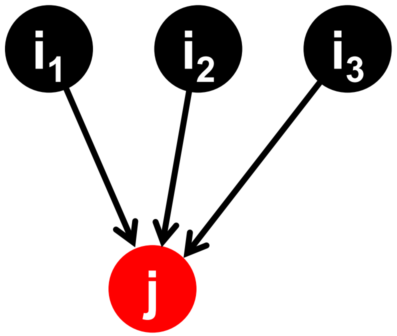
$$\mathbf{r_y} = \mathbf{r_y}/2 + \mathbf{r_a}/2$$
$$\mathbf{r_a} = \mathbf{r_y}/2 + \mathbf{r_m}$$
$$\mathbf{r_m} = \mathbf{r_a}/2$$

- **Example:**

r_y		1/3	1/3	5/12	9/24		6/15
r_a	=	1/3	3/6	1/3	11/24	…	6/15
r_m		1/3	1/6	3/12	1/6		3/15

Iteration 0, 1, 2, …

Random Walk Interpretation

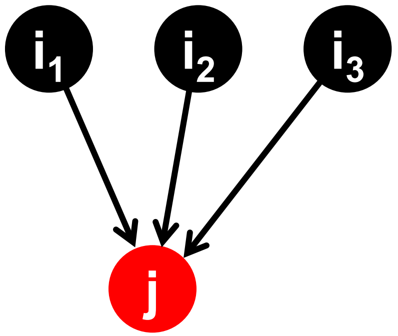


$$r_j = \sum_{i \to j} \frac{r_i}{\mathrm{d}_{\mathrm{out}}(\mathrm{i})}$$

- **Imagine a random web surfer:**
  - At any time $\boldsymbol{t}$, surfer is on some page $\boldsymbol{i}$
  - At time $\boldsymbol{t+1}$, 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:**
  - $\boldsymbol{p(t)}$ … vector whose $\boldsymbol{i}^{\text{th}}$ coordinate is the prob. that the surfer is at page $\boldsymbol{i}$ at time $\boldsymbol{t}$
  - So, $\boldsymbol{p(t)}$ is a probability distribution over pages

The Stationary Distribution



$$p(t+1) = \mathbf{M} \cdot p(t)$$

- **Where is the surfer at time $t+1$?**
  - Follows a link uniformly at random

$$\boldsymbol{p(t+1) = M \cdot p(t)}$$

- Suppose the random walk reaches a steady state

$$\boldsymbol{p(t+1) = M \cdot p(t) = p(t)}$$

  - then $\boldsymbol{p(t)}$ is **stationary distribution** of a random walk

- **Our original rank vector** $\boldsymbol{r}$ satisfies $\boldsymbol{r = M \cdot r}$
  - **So, $\boldsymbol{r}$ is a stationary distribution for the random walk**

Existence and Uniqueness

- **A central result from the theory of random walks (a.k.a. Markov processes):**

For graphs that satisfy **certain conditions**, the **stationary distribution is unique** and eventually will be reached no matter what the initial probability distribution at time **t = 0**

PageRank: Google Formulation

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

or equivalently

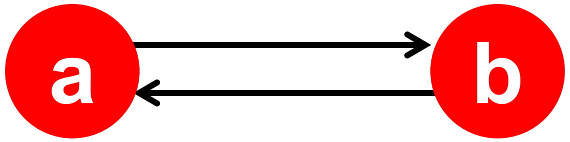
$$r = Mr$$

Google's PageRank with 3 Questions:

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

1. Does this converge?

The "Spider trap" problem



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

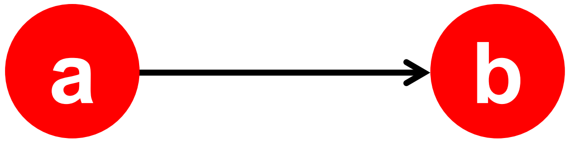
- **Example:**

r_a	1	0	1	0
r_b	0	1	0	1

Iteration 0, 1, 2, …

2. Does this converge to what we want?

The "Dead end" problem



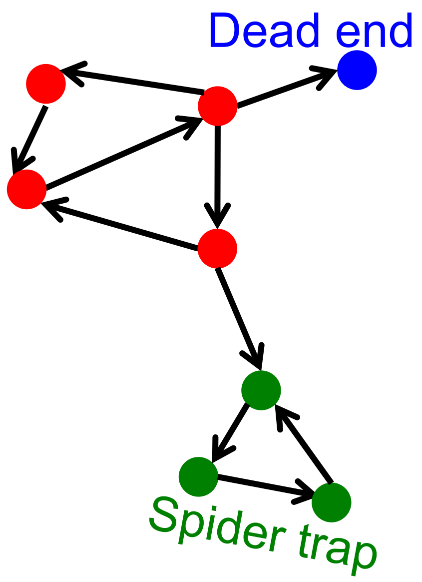
$$r_j^{(t+1)} = \sum_{i \to j} \frac{r_i^{(t)}}{\mathrm{d}_i}$$

- **Example:**

r_a	1	0	0	0
r_b	0	1	0	0

Iteration 0, 1, 2, …

PageRank: Problems

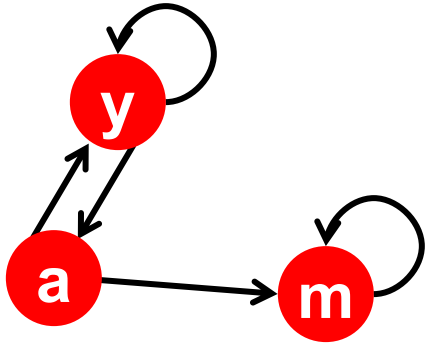


2 problems:

- **(1)** Some pages are **dead ends** (have no out-links)
  - Random walk has “nowhere” to go to
  - Such pages cause importance to “leak out”
- **(2)** **Spider traps:** (all out-links are within the group)
  - Random walker gets “stuck” in a trap
  - And eventually spider traps absorb all importance

Problem: Spider Traps

- **Power Iteration:**
  - Set $r_j = 1$
  - $r_j = \sum_{i \to j} \frac{r_i}{d_i}$
    - And iterate



m is a spider trap

	y	a	m
y	½	½	0
a	½	0	0
m	0	½	1

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

- **Example:**

$$\begin{array}{cccccccc} r_y & & 1/3 & 2/6 & 3/12 & 5/24 & & 0 \\ r_a & = & 1/3 & 1/6 & 2/12 & 3/24 & \ldots & 0 \\ r_m & & 1/3 & 3/6 & 7/12 & 16/24 & & 1 \end{array}$$

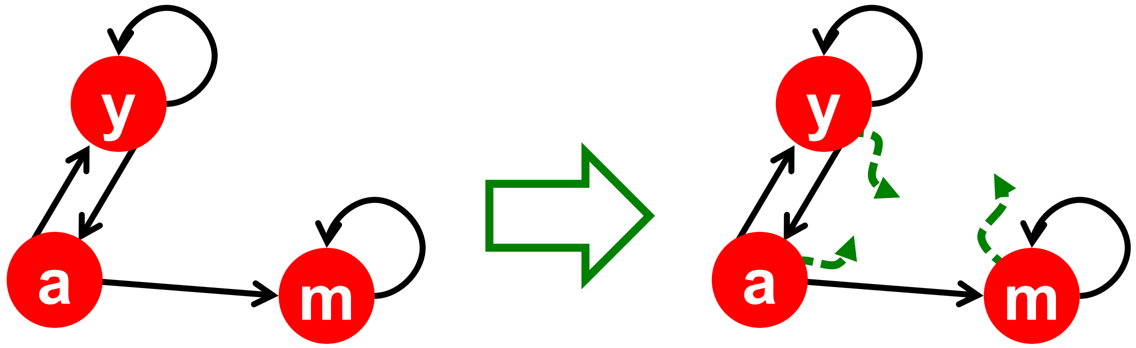
Iteration 0, 1, 2, …

All the PageRank score gets “trapped” in node m.

Solution: Teleports!

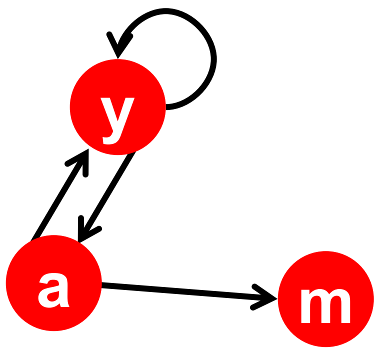
- **The Google solution for spider traps: At each time step, the random surfer has two options**
  - With prob. β, follow a link at random
  - With prob. **1-**β, jump to some random page
  - Common values for β are in the range 0.8 to 0.9

- **Surfer will teleport out of spider trap within a few time steps**



Problem: Dead Ends

- **Power Iteration:**
  - Set $r_j = 1$
  - $r_j = \sum_{i \to j} \frac{r_i}{d_i}$
    - And iterate



$$\begin{array}{c|ccc|} & y & a & m \\ \hline y & ½ & ½ & 0 \\ a & ½ & 0 & 0 \\ m & 0 & ½ & 0 \\ \hline \end{array}$$

$r_y = r_y/2 + r_a/2$

$r_a = r_y/2$

$r_m = r_a/2$

- **Example:**

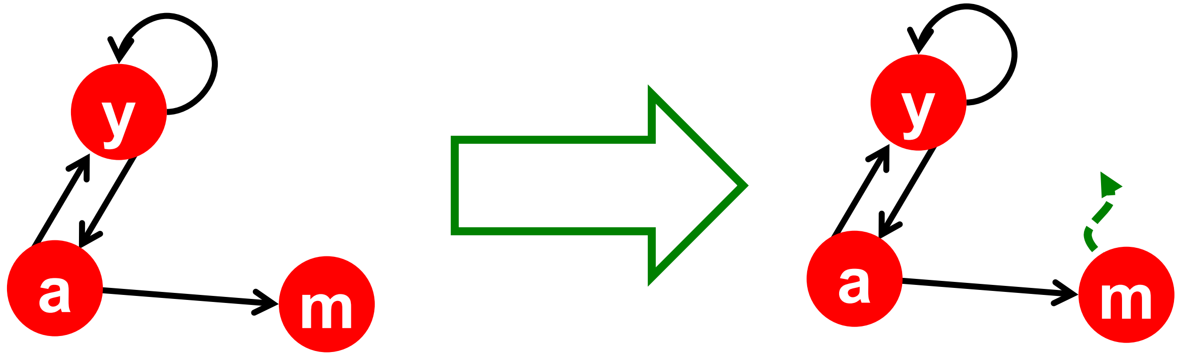
$$\begin{matrix} r_y & & 1/3 & 2/6 & 3/12 & 5/24 & & 0 \\ r_a & = & 1/3 & 1/6 & 2/12 & 3/24 & \dots & 0 \\ r_m & & 1/3 & 1/6 & 1/12 & 2/24 & & 0 \end{matrix}$$

Iteration 0, 1, 2, …

Here the PageRank “leaks” out since the matrix is not stochastic.

Solution: Always Teleport!

- **Teleports:** Follow random teleport links with 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

- **Google's solution that does it all:**
  At each step, random surfer has two options:
  - With probability β, follow a link at random
  - With probability $1\text{-}\beta$, jump to some random page
- **PageRank equation** [Brin-Page, 98]

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

d_i … out-degree of node i

This formulation assumes that M has no dead ends. We can either preprocess matrix M to remove all dead ends or explicitly follow random teleport links with probability 1.0 from dead-ends.

The Google Matrix

- **PageRank equation**

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

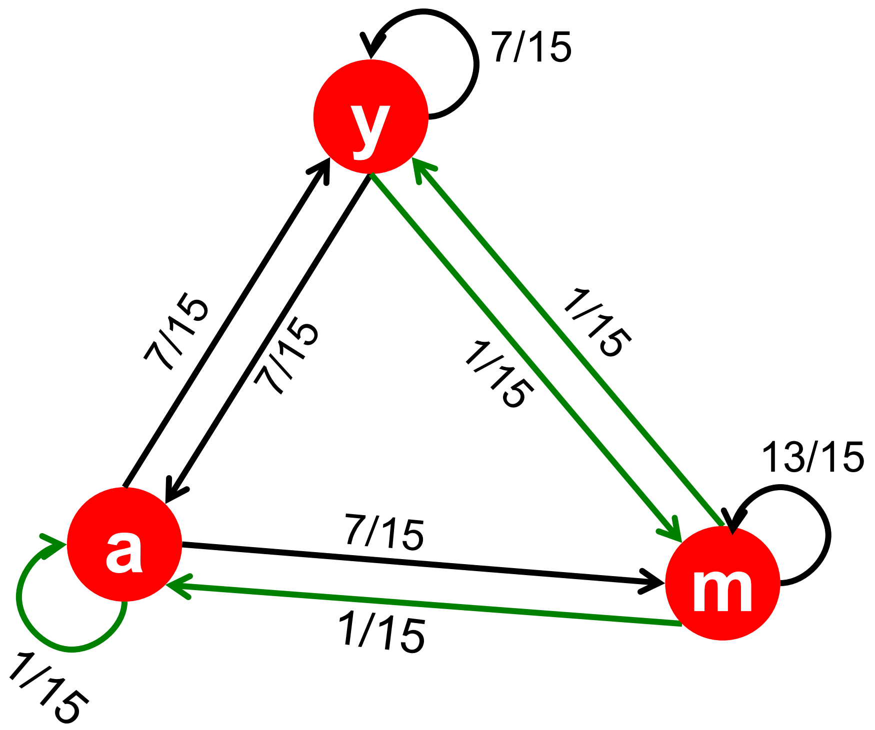
- **The Google Matrix *A*:**

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

$[1/N]_{NxN}$…N by N matrix where all entries are 1/N

- **We have a recursive problem: $\boldsymbol{r} = \boldsymbol{A} \cdot \boldsymbol{r}$**
  And the Power method still works!
- **What is β?**
  - In practice $\beta = 0.8, 0.9$ (make 5 steps on avg., jump)

Random Teleports (β=0.8)



$$0.8 \overset{\textbf{M}}{\boxed{\begin{array}{ccc} 1/2 & 1/2 & 0 \\ 1/2 & 0 & 0 \\ 0 & 1/2 & 1 \end{array}}} + 0.2 \overset{\textbf{[1/N]}_{\textbf{NxN}}}{\boxed{\begin{array}{ccc} 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \end{array}}}$$

$$\begin{array}{c} y \\ a \\ m \end{array} \underset{\textbf{A}}{\boxed{\begin{array}{ccc} 7/15 & 7/15 & 1/15 \\ 7/15 & 1/15 & 1/15 \\ 1/15 & 7/15 & 13/15 \end{array}}}$$

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

How do we actually compute PageRank?

- **Key step is matrix-vector multiplication**
  - $\boldsymbol{r}^{new} = \boldsymbol{A} \cdot \boldsymbol{r}^{old}$

- Easy if we have enough main memory to hold $\mathbf{A}$, $\mathbf{r}^{old}$, $\mathbf{r}^{new}$

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

- **Say N = 1 billion pages**
  - We need 4 bytes for each entry (say)
  - 2 billion entries for vectors, approx 8GB
  - Matrix **A** has N^2 entries
    - 10^{18} is a large number!

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

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

Matrix Formulation

- Suppose there are $\boldsymbol{N}$ pages
- Consider page $\boldsymbol{i}$, with $\mathbf{d}_i$ out-links
- We have $\boldsymbol{M_{ji} = 1/|d_i|}$ when $\boldsymbol{i \rightarrow j}$
  and $\boldsymbol{M_{ji} = 0}$ otherwise
- **The random teleport is equivalent to:**
  - Adding a **teleport link** from $\boldsymbol{i}$ to every other page and setting transition probability to $\boldsymbol{(1-\beta)/N}$
  - Reducing the probability of following each out-link from $\boldsymbol{1/|d_i|}$ to $\boldsymbol{\beta/|d_i|}$
  - **Equivalent:** Tax each page a fraction $\boldsymbol{(1-\beta)}$ of its score and redistribute evenly

Rearranging the Equation

- $\boldsymbol{r} \;\; = \boldsymbol{A} \cdot \boldsymbol{r},$ **where** $\boldsymbol{A_{ji} = \beta\, M_{ji} + \frac{1-\beta}{N}}$
- $r_j \;\; = \sum_{\mathrm{i}=1}^{N} A_{ji} \cdot r_i$
- $r_j \;\; = \sum_{i=1}^{N} \left[\beta\, M_{ji} + \frac{1-\beta}{N}\right] \cdot r_i$

$$= \sum_{\mathrm{i}=1}^{N} \beta\, M_{ji} \cdot r_i + \frac{1-\beta}{N} \sum_{\mathrm{i}=1}^{N} r_i$$

$$= \sum_{\mathrm{i}=1}^{N} \beta\, M_{ji} \cdot r_i + \frac{1-\beta}{N} \qquad \text{since } \sum r_i = 1$$

- **So we get:** $\boldsymbol{r = \beta\, M \cdot r + \left[\frac{1-\beta}{N}\right]_N}$

Note: Here we assumed **M** has no dead-ends

$[x]_N$ … a vector of length *N* with all entries *x*

Sparse Matrix Formulation

- We just rearranged the **PageRank equation**
  - $$\boldsymbol{r} = \boldsymbol{\beta M} \cdot \boldsymbol{r} + \left[\frac{\mathbf{1}-\boldsymbol{\beta}}{\boldsymbol{N}}\right]_{\boldsymbol{N}}$$
    - where **[(1-β)/N]$_N$** is a vector with all ***N*** entries **(1-β)/N**

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

PageRank: The Complete Algorithm

- **Input: Graph G and parameter β**
  - Directed graph G (can have **spider traps** and **dead ends**)
  - Parameter β
- **Output: PageRank vector r^{new}**
  - **Set:** $r_j^{old} = \frac{1}{N}$
  - **repeat until convergence:** $\sum_j \left| r_j^{new} - r_j^{old} \right| > \varepsilon$
    - $\forall j$: $\boldsymbol{r'}_j^{new} = \sum_{i \to j} \boldsymbol{\beta} \frac{\boldsymbol{r}_i^{old}}{\boldsymbol{d}_i}$

      $\boldsymbol{r'}_j^{new} = \mathbf{0}$ if in-degree of $\boldsymbol{j}$ is **0**
    - **Now re-insert the leaked PageRank:**

      $\forall \boldsymbol{j}$: $\boldsymbol{r}_j^{new} = \boldsymbol{r'}_j^{new} + \frac{\mathbf{1} - \boldsymbol{S}}{\boldsymbol{N}}$ **Where:** $S = \sum_j r_j'^{new}$
    - $\boldsymbol{r}^{old} = \boldsymbol{r}^{new}$

If the graph has no dead-ends then the amount of leaked PageRank is **1-β**. But since we have dead-ends the amount of leaked PageRank may be larger. We have to explicitly account for it by computing **S**.

Sparse Matrix Encoding

- **Encode sparse matrix using only nonzero entries**
  - Space proportional roughly to number of links
  - Say 10B, or 4*10*1 billion = 40GB
  - **Still won't fit in memory, but will fit on disk**

source node	degree	destination nodes
0	3	1, 5, 7
1	5	17, 64, 113, 117, 245
2	2	13, 23

Basic Algorithm: Update Step

- **Assume enough RAM to fit r^{new} into memory**
  - Store r^{old} and matrix **M** on disk
- **1 step of power-iteration is:**

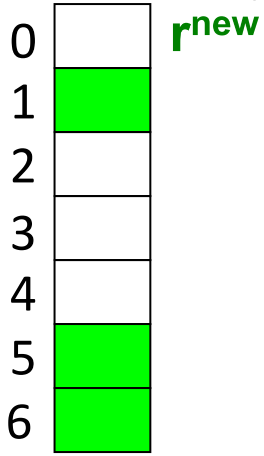
  Initialize all entries of $\mathbf{r^{new}} = \mathbf{(1\text{-}\beta) / N}$

  For each page $\boldsymbol{i}$ (of out-degree $\boldsymbol{d_i}$):

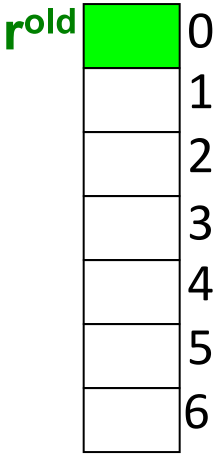
  Read into memory: $\boldsymbol{i}$, $\boldsymbol{d_i}$, $\boldsymbol{dest_1, \ldots, dest_{d_i}, r^{old}(i)}$

  For $\mathbf{j = 1 \ldots d_i}$

  $\mathbf{r^{new}(dest_j) \mathrel{+}= \beta\, r^{old}(i) / d_i}$



source	degree	destination
0	3	1, 5, 6
1	4	17, 64, 113, 117
2	2	13, 23



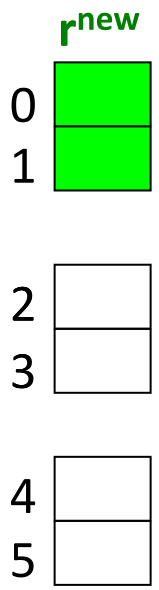
Analysis

- **Assume enough RAM to fit r^{new} into memory**
  - Store $\boldsymbol{r}^{old}$ and matrix $\boldsymbol{M}$ on disk

- **In each iteration, we have to:**
  - Read $\boldsymbol{r}^{old}$ and $\boldsymbol{M}$
  - Write $\boldsymbol{r}^{new}$ back to disk
  - **Cost per iteration of Power method:**
    $= 2|\boldsymbol{r}| + |\boldsymbol{M}|$

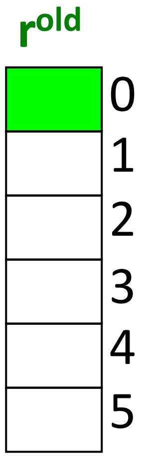
- **Question:**
  - What if we could not even fit $\boldsymbol{r}^{new}$ in memory?

Block-based Update Algorithm



src	degree	destination
0	4	0, 1, 3, 5
1	2	0, 5
2	2	3, 4

M



- Break $\boldsymbol{r}^{new}$ into $\boldsymbol{k}$ blocks that fit in memory
- Scan $\boldsymbol{M}$ and $\boldsymbol{r}^{old}$ once for each block

Analysis of Block Update

- **Similar to nested-loop join in databases**
  - Break $\boldsymbol{r}^{\text{new}}$ into $\boldsymbol{k}$ blocks that fit in memory
  - Scan $\boldsymbol{M}$ and $\boldsymbol{r}^{\text{old}}$ once for each block

- **Total cost:**
  - $\boldsymbol{k}$ scans of $\boldsymbol{M}$ and $\boldsymbol{r}^{\text{old}}$
  - **Cost per iteration of Power method:**
    $k(|\boldsymbol{M}| + |\boldsymbol{r}|) + |\boldsymbol{r}| = \boldsymbol{k|M| + (k+1)|r|}$

- **Can we do better?**
  - **Hint:** $\boldsymbol{M}$ is much bigger than $\boldsymbol{r}$ (approx 10-20x), so we must avoid reading it $\boldsymbol{k}$ times per iteration

Block-Stripe Update Algorithm

r^{new}

0		
1		

src	degree	destination
0	4	0, 1
1	3	0
2	2	1

2		
3		

0	4	3
2	2	3

4		
5		

0	4	5
1	3	5
2	2	4

r^{old}

	0	
	1	
	2	
	3	
	4	
	5	

Break *M* into stripes! Each stripe contains only destination nodes in the corresponding block of r^{new}

Block-Stripe Analysis

- Break $\boldsymbol{M}$ into stripes
  - Each stripe contains only destination nodes in the corresponding block of $\boldsymbol{r}^{\text{new}}$
- Some additional overhead per stripe
  - But it is usually worth it
- **Cost per iteration of Power method =$|M|(1+\varepsilon) + (k+1)|r|$**

Some Problems with PageRank

- **Measures generic popularity of a page**
  - Biased against topic-specific authorities
  - **Solution:** Topic-Specific PageRank (**next**)
- **Uses a single measure of importance**
  - Other models of importance
  - **Solution:** Hubs-and-Authorities
- **Susceptible to Link spam**
  - Artificial link topographies created in order to boost page rank
  - **Solution:** TrustRank

Questions???

Acknowledgements

Most of this lecture slides are obtained from the Mining Massive Datasets course: http://www.mmds.org/