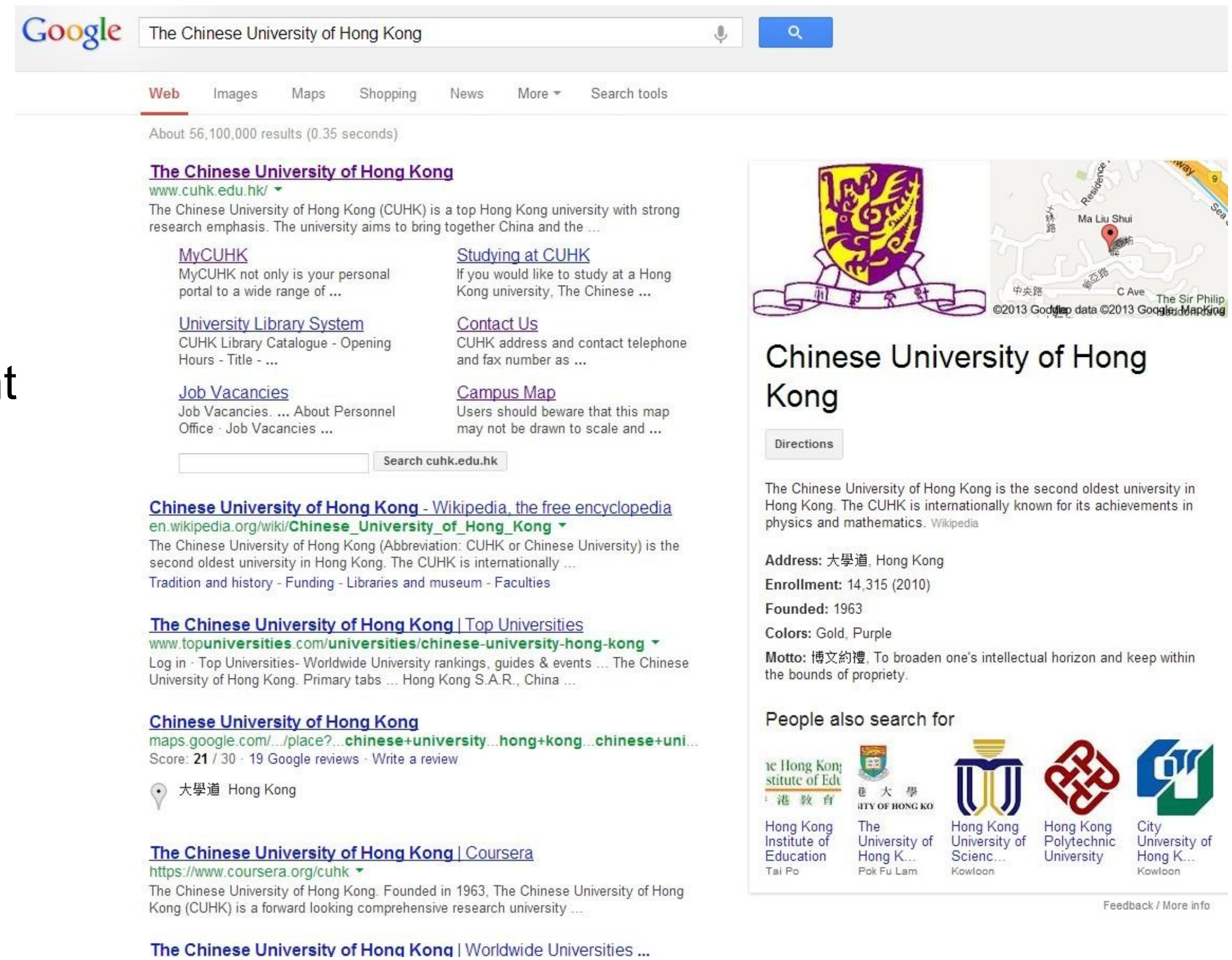
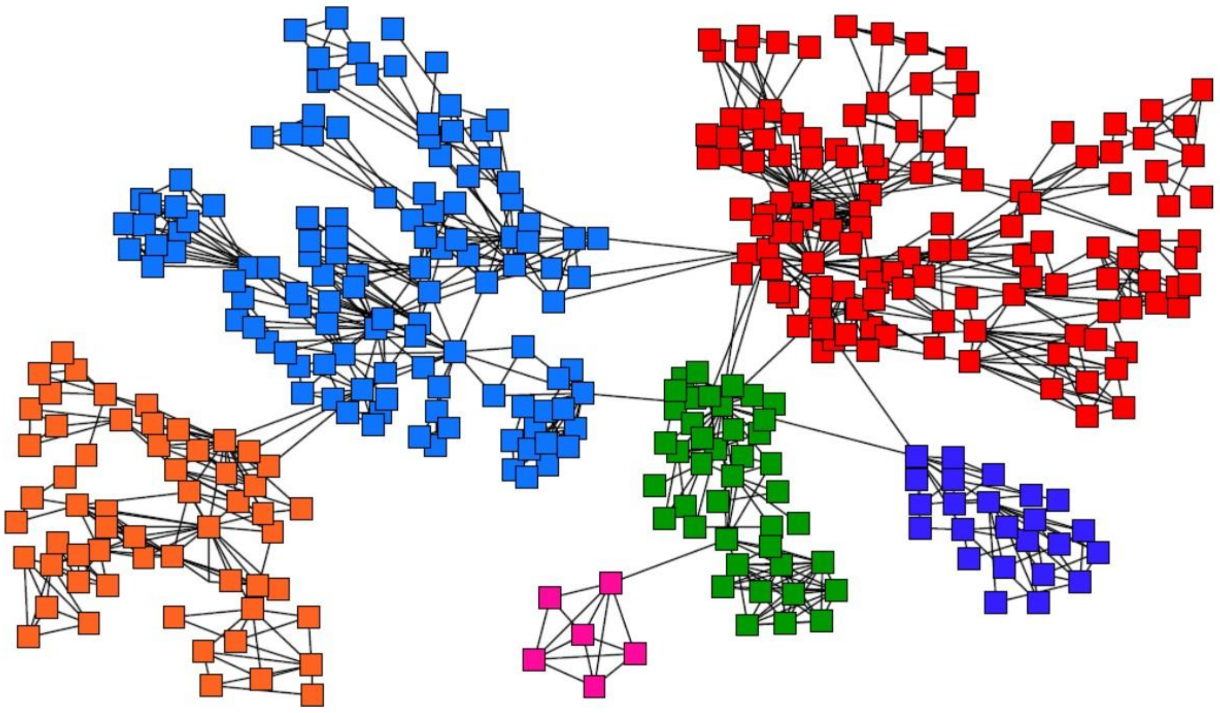
# Massive Link Analysis

# What's the Mechanism Behind Google?

How google return such kind of rankings (e.g. cuhk homepage first then Wikipedia page)? One important factor is PageRank score.



# How to make PageRank computation scalable



- There are billions of web pages and hyperlinks between them, how to compute their ranking score (e.g., PageRank) efficiently?

# Outline

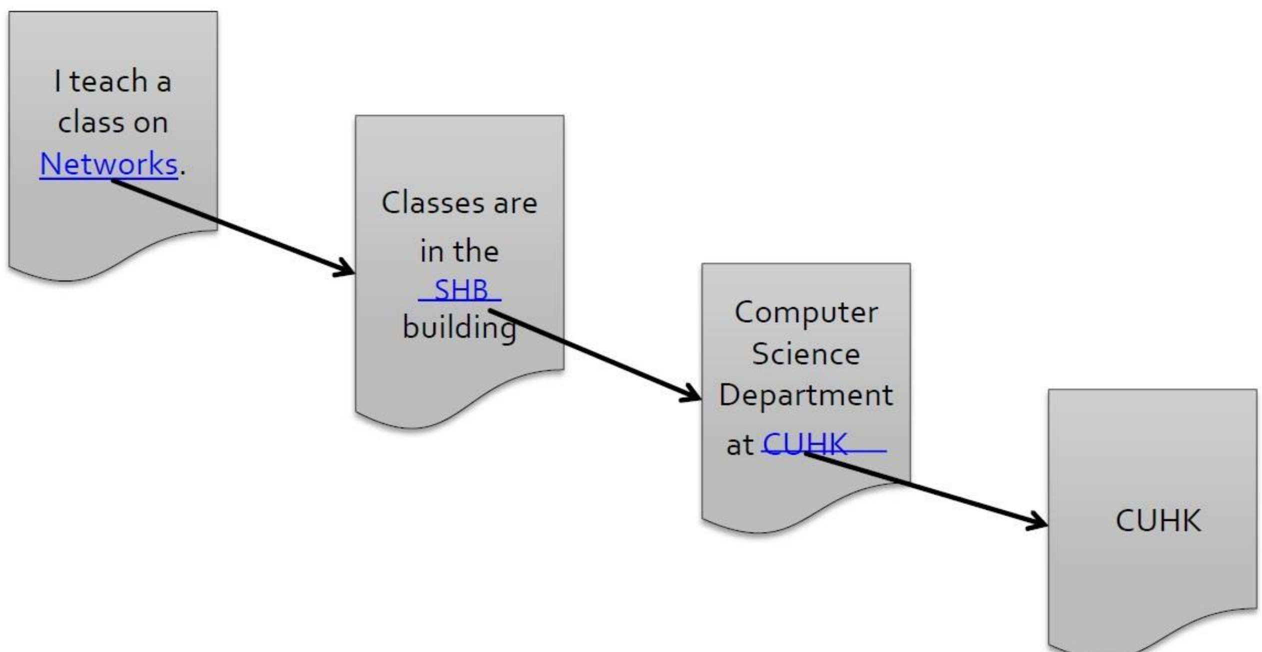
- Web as a Graph
- PageRank
- Topic-Specific PageRank
- Trust-Rank
- Further Reading

# Outline

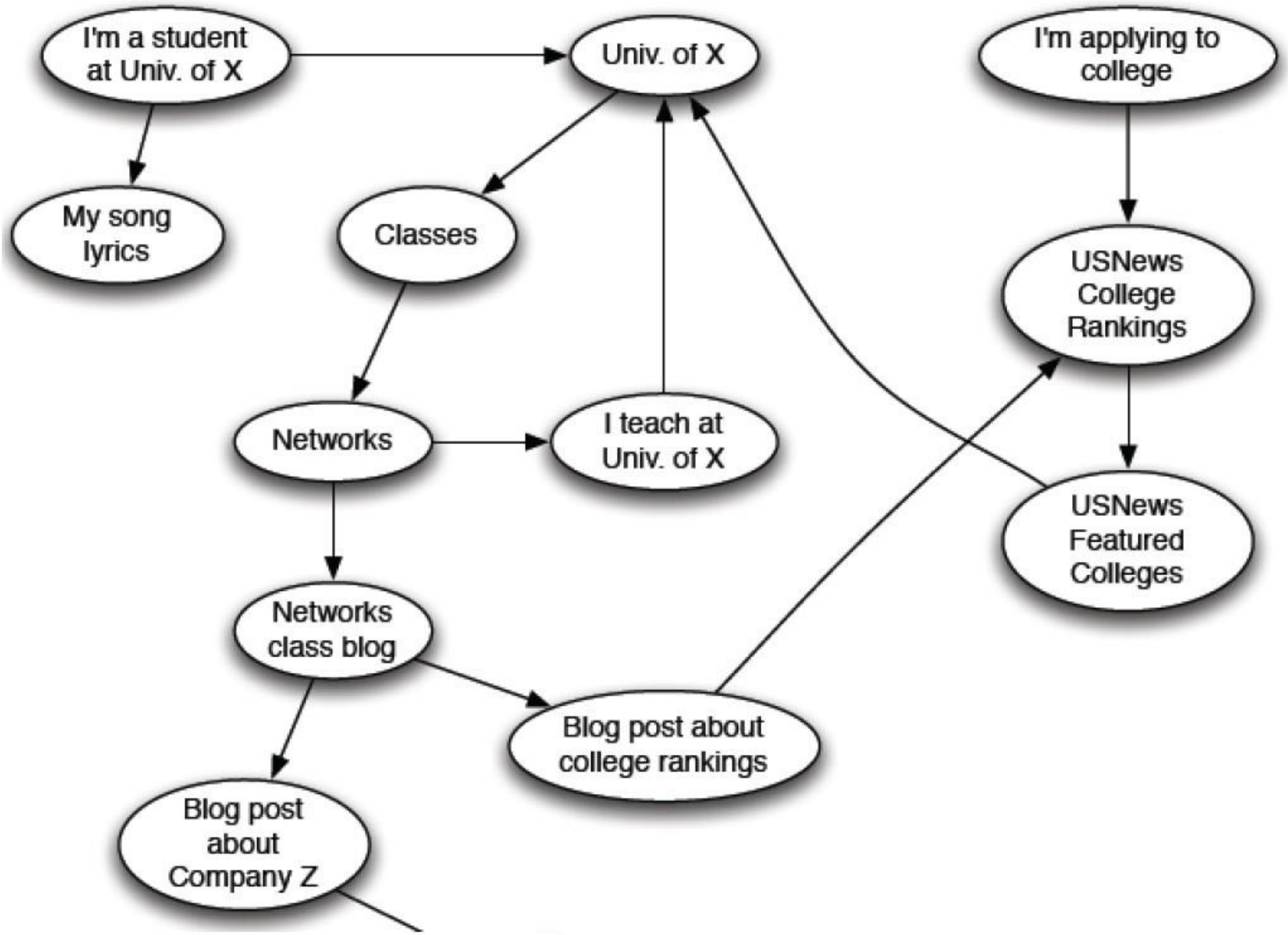
- Web as a Graph
- PageRank
- Topic-Specific PageRank
- Trust-Rank
- Further Reading

# Web as a Graph

- Web as a directed graph:
  - Nodes: Web pages
  - Edges: Hyperlinks



# Web as a Directed Graph



# Broad Question

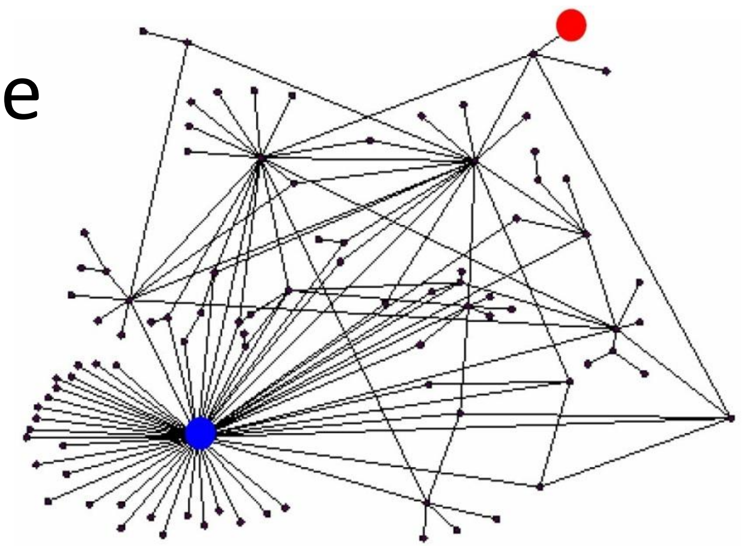
- How to organize the Web?
- First try: Human curated Web directories
  - Yahoo, DMOZ, LookSmart
- Second try: Web Search
  - Information Retrieval investigates: Find relevant docs in a small and trusted set
    - Newspaper articles, Patents, etc.
  - But: Web is huge, full of untrusted documents, random things, web spam, etc.

# Web Search: 2 Challenges

- Web contains many sources of information: Who to “trust”?
  - Trick: Trustworthy pages may point to each other
- 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"
  - www.cuhk.edu.hk vs. www.joe-schmoe.com
- There is large diversity in the web-graph node connectivity.
  - 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
  – PageRank
  – Topic-Specific (Personalized) PageRank
  – Web Spam Detection Algorithms, e.g. TrustRank

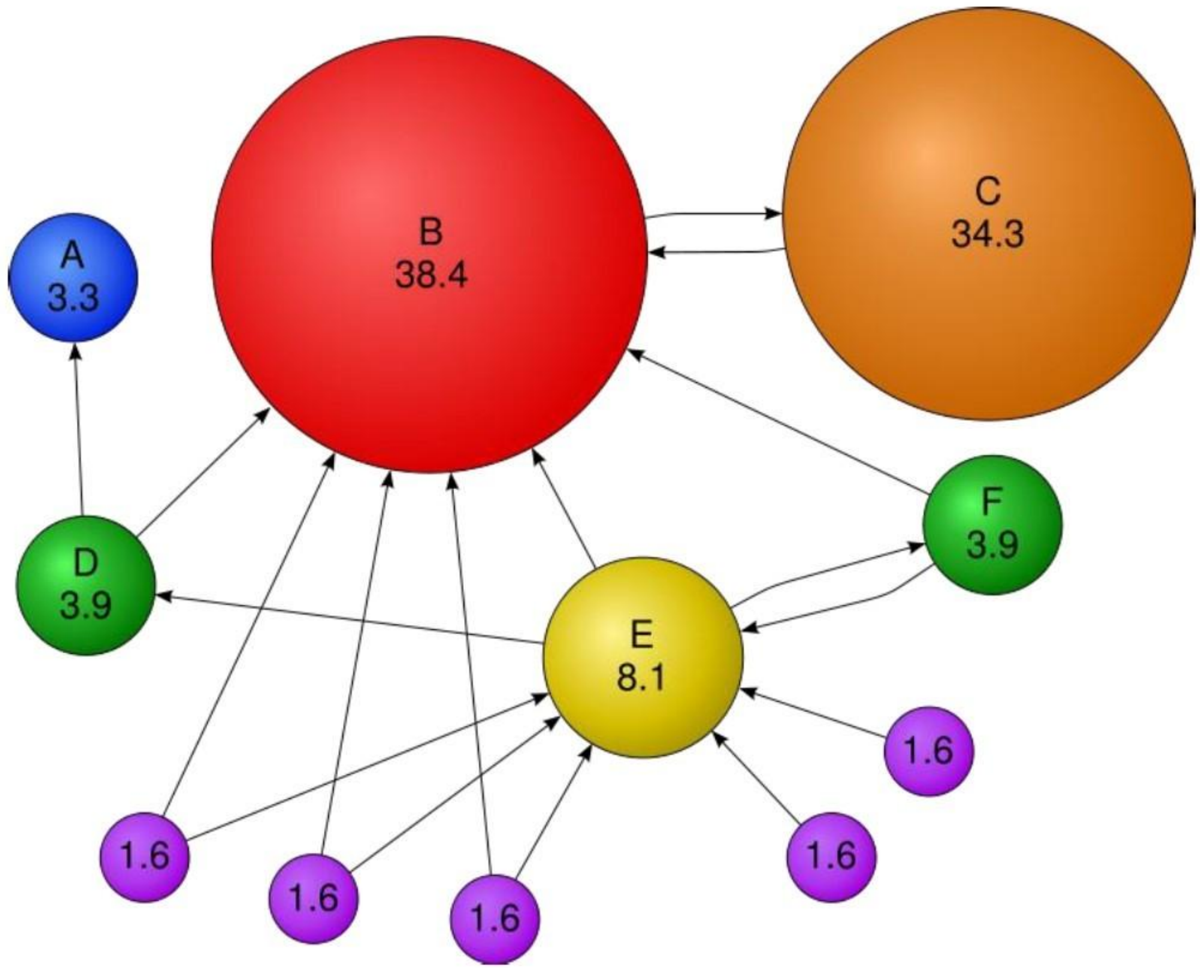
# Outline

- Web as a Graph
- PageRank
- Topic-Specific PageRank
- Trust-Rank
- Further Reading

# 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.cuhk.edu.hk has 15,432 in-links
  - www.joe-schmoe.com has 1 in-link
- Are all in-links equal?
  - Link from important pages count more
  - Recursive question!

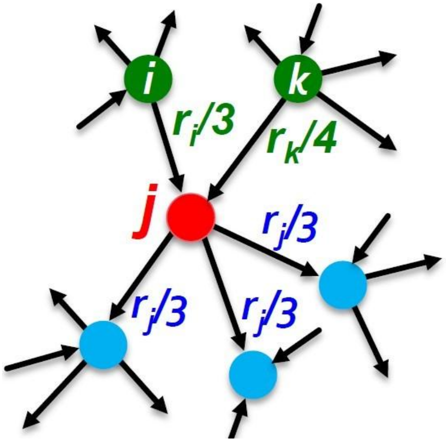
# Example: PageRank Scores



# 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

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

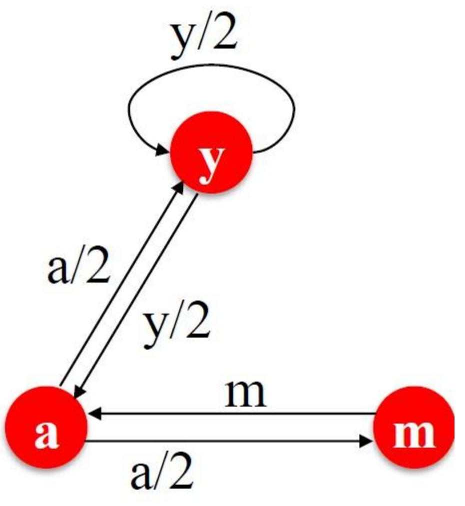


# PageRank: The “Flow” Model

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

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

$d_i \ldots$ out-degress of node $i$




“Flow” equations:

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

# Solving the Flow Equations

- 3 equations, 3 unknowns, no constants
  - No unique solution
  - All solutions equivalent modulo the scale factor
- Additional constraint forces uniqueness:
  - $r_y + r_a + r_m = 1$
  - Solution: $r_y = \frac{2}{5}, r_a = \frac{2}{5}, r_m = \frac{1}{5}$

**Flow equations:**

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

- Gaussian elimination method works for small examples, but we need a better method for large web-size graphs

# PageRank: Matrix Formulation

- Stochastic adjacency matrix M
  - Let page $i$ have $d_i$ out-links
  - If $i \to j$, then $M_{ji} = \frac{1}{d_i}$, else $M_{ji} = 0$
    - $M$ is a column stochastic matrix, i.e., columns sum to 1
- Rank vector 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 $r_j = \sum_{i \to j} \frac{r_i}{d_i}$

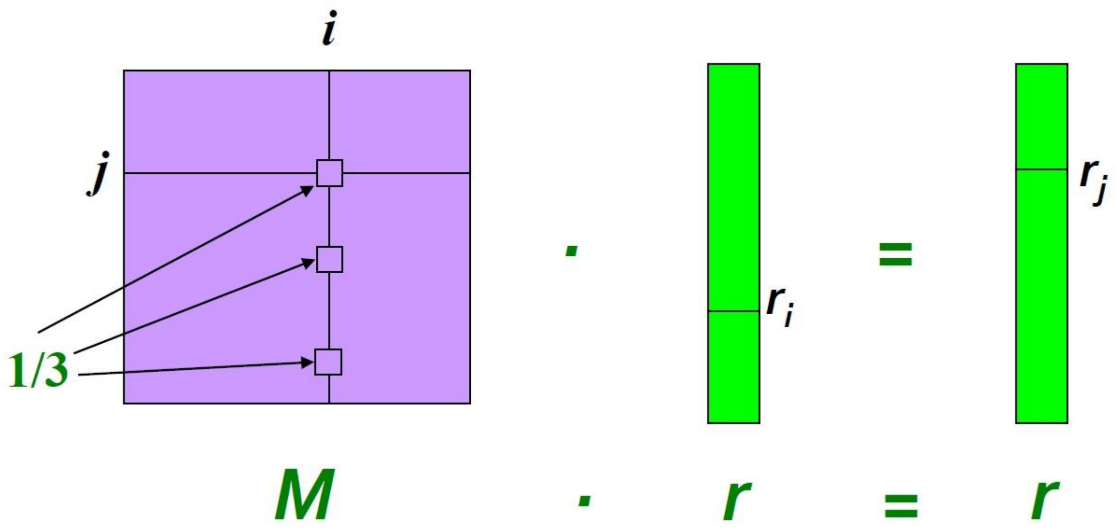
$$r = M \cdot r$$

# Example

- Remember the flow equation: $r_j = \sum_{i \to j} \frac{r_i}{d_i}$
- Flow equation in the matrix form:

$$r = M \cdot r$$

  – Suppose page $i$ links to 3 pages, including $j$



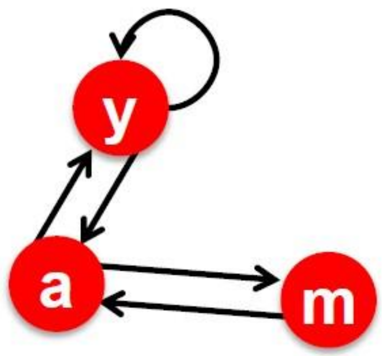
# Eigenvector Formulation

- The flow equations can be written $r = M \cdot r$
- So the rank vector $\boldsymbol{r}$ is an eigenvector of the stochastic web matrix $\boldsymbol{M}$
  - In fact, its first or principal eigenvector, with corresponding eigenvalue **1**
    - Largest eigenvalue of $\boldsymbol{M}$ is **1** since $\boldsymbol{M}$ is column stochastic
- We can now efficiently solve for r!
  - The method is called Power iteration.

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

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

# Example: Flow Equation & *M*




| | y | a | m |
|---|---|---|---|
| **y** | ½ | ½ | 0 |
| **a** | ½ | 0 | 1 |
| **m** | 0 | ½ | 0 |

$$r = M \cdot r$$

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

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

# 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: $\mathbf{r}^{(0)} = [1/N, \ldots, 1/N]^T$
  - Iterate: $\mathbf{r}^{(t+1)} = \mathbf{M} \cdot \mathbf{r}^{(t)}$
  - Stop when $|\mathbf{r}^{(t+1)} - \mathbf{r}^{(t)}|_1 < \epsilon$
    - $|\mathbf{x}|_1 = \sum_{1 \leq i \leq N} |x_i|$ is the L1 norm

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

$d_i \ldots$ out-degress of node $i$

# Why Power Iteration Works?

- Power iteration:
  - A method for finding dominant eigenvector (the vector corresponding to the largest eigenvalue)
    - $\mathbf{r}^{(1)} = \mathbf{M} \cdot \mathbf{r}^{(0)}$
    - $\mathbf{r}^{(2)} = \mathbf{M} \cdot \mathbf{r}^{(1)} = \mathbf{M}(\mathbf{M}\mathbf{r}^{(1)}) = \mathbf{M}^{\mathbf{2}} \cdot \mathbf{r}^{(0)}$

      $\mathbf{r}^{(3)} = \mathbf{M} \cdot \mathbf{r}^{(2)} = \mathbf{M}(\mathbf{M}^{\mathbf{2}}\mathbf{r}^{(0)}) = \mathbf{M}^{\mathbf{3}} \cdot \mathbf{r}^{(0)}$
- Claim:
  - Sequence $\mathbf{M} \cdot \mathbf{r}^{(0)}, \mathbf{M}^{\mathbf{2}} \cdot \mathbf{r}^{(0)}, \ldots, \mathbf{M}^{\mathbf{k}} \cdot \mathbf{r}^{(0)}, \ldots$ approaches the dominant eigenvector of ***M***

# Why Power Iteration Works?

- Proof:
  - Assume ***M*** has ***n*** linearly independent eigenvectors $x_1, x_2, \ldots, x_n$ with corresponding eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_n$, where $\lambda_1 > \lambda_2 > \ldots > \lambda_n$
  - Vectors $x_1, x_2, \ldots, x_n$ form a basis and thus we can write: $r^{(0)} = c_1 x_1 + c_2 x_2 + \ldots + c_n x_n$
  - $$\begin{aligned} Mr^{(0)} &= M(c_1 x_1 + c_2 x_2 + \ldots + c_n x_n) \\ &= c_1(Mx_1) + c_2(Mx_2) + \ldots + c_n(Mx_n) \\ &= c_1(\lambda_1 x_1) + c_2(\lambda_2 x_2) + \ldots + c_n(\lambda_n x_n) \end{aligned}$$
  - Repeated multiplication on both sides:

$$M^k r^{(0)} = c_1(\lambda_1^k x_1) + c_2(\lambda_2^k x_2) + \ldots + c_n(\lambda_n^k x_n)$$

# Why Power Iteration Works?

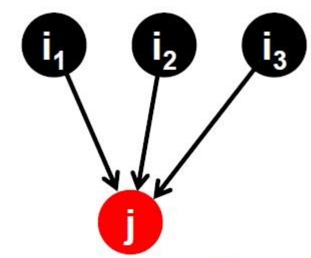
- Proof: (cont.)
  - Repeated multiplication on both sides produces

$$M^k r^{(0)} = \lambda_1^k \left[ c_1 x_1 + c_2 \left( \frac{\lambda_2}{\lambda_1} \right)^k x_2 + \ldots + c_n \left( \frac{\lambda_2}{\lambda_1} \right)^k x_n \right]$$

  - Since $\lambda_1 > \lambda_2$ then $\frac{\lambda_2}{\lambda_1}, \frac{\lambda_3}{\lambda_1} \ldots < 1$, so $\frac{\lambda_i}{\lambda_1} = 0 \text{ as } k \to \infty$
  - Thus $M^k r^{(0)} \approx c_1(\lambda_1^k x_1)$
    - Note if $c_1 = 0$, then the method won't converge

# Random Walk Interpretation

- Imagine a random web surfer:
  - At any time t, surfer is on some page *i*
  - At time t+1, the surfer follows an out-link from *i* uniformly at random
  - Ends up on some page *j* linked from *i*
  - Process repeats indefinitely




$$r_j = \sum_{i \to j} \frac{r_i}{d_{out}(i)}$$

- Let:
  - p(t) … vector whose $i^{th}$ coordinate is the prob. that the surfer is at page *i* at time *t*
  - So p(t) is a probability distribution over pages

# The Stationary Distribution

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

    $$p(t+1) = M \cdot p(t)$$

- Suppose the random walk reaches a state

  $$p(t+1) = M \cdot p(t) = p(t)$$

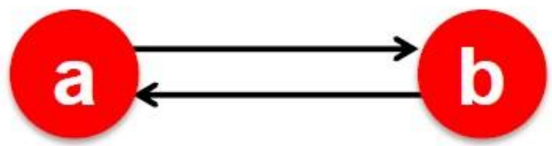
  Then p(t) is stationary distribution of a random walk

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

# PageRank

- Three questions:
  – Does this converge?
  – Does it converge to what we want?
  – Are results reasonable?

# Does This Converge?



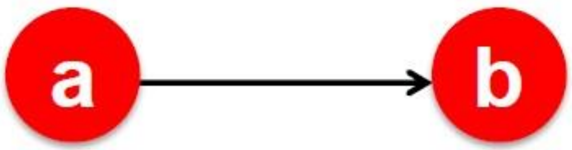

$$r_j^{(t+1)} = \sum_{i \to j} \frac{r_i^{(t)}}{d_i}$$

- **Example:**

$$\begin{array}{ccccc} r_a & & 1 & 0 & 1 & 0 \\ & = & & & & \\ r_b & & 0 & 1 & 0 & 1 \end{array}$$

Iteration 0, 1, 2, …

# Does it Converge to What We Want?




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

- **Example:**

$$\begin{matrix} r_a \\ r_b \end{matrix} = \begin{matrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{matrix}$$

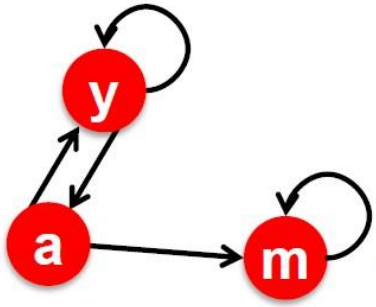
Iteration 0, 1, 2, …

# PageRank: Problems

- Two problems:
  - Spider traps: all out-links are within the group
    - Eventually spider traps absorb all importance
  - Some pages are dead ends (have no out-links)
    - Such pages cause importance to “leak out”

# Problem: Spider Traps

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




| | y | a | m |
|---|---|---|---|
| y | ½ | ½ | 0 |
| a | ½ | 0 | 0 |
| m | 0 | ½ | 1 |

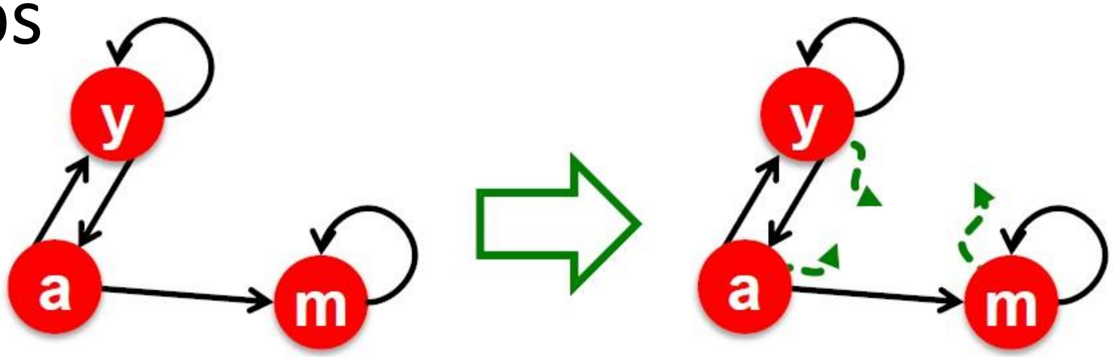
$$r_y = r_y/2 + r_a/2$$
$$r_a = r_y/2$$
$$r_m = r_a/2 + r_m$$

$$\begin{bmatrix} r_y \\ r_a \\ r_m \end{bmatrix} = \begin{matrix} 1/3 & 2/6 & 3/12 & 5/24 & & 0 \\ 1/3 & 1/6 & 2/12 & 3/24 & \dots & 0 \\ 1/3 & 3/6 & 7/12 & 16/24 & & 1 \end{matrix}$$

Iteration 0, 1, 2, …

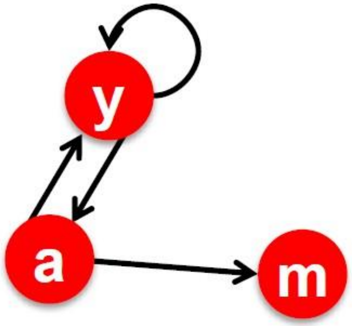
# Solution: Random Teleports

- The Google solution for spider traps: At each time step, the random surfer has two options
  - With prob. $\beta$, follow a link at random
  - With prob. $1-\beta$ , jump to some random page
  - Common values for $\beta$ 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
- Example




| | y | a | m |
|---|---|---|---|
| y | ½ | ½ | 0 |
| a | ½ | 0 | 0 |
| m | 0 | ½ | 0 |

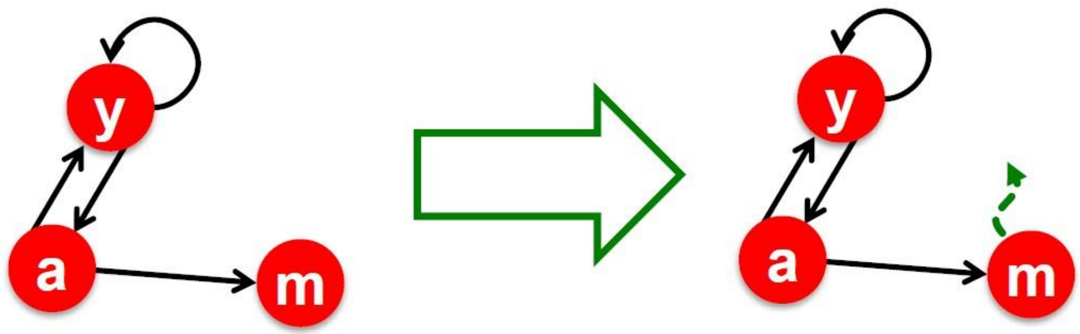
$$r_y = r_y/2 + r_a/2$$
$$r_a = r_y/2$$
$$r_m = r_a/2$$

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

Iteration 0, 1, 2, …

# 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?

$$\mathbf{r^{(t+1)}} = Mr^{(t)}$$

- Markov chains
  - Set of states **X**
  - Transition matrix **P** where $P_{ij} = P(X_t = i | X_{t-1} = j)$
  - $\pi$ specifying the stationary probability of being at each state $x \in X$
  - Goal is to find $\pi$ such that $\pi = P\pi$

# Why Is This Analogy Useful?

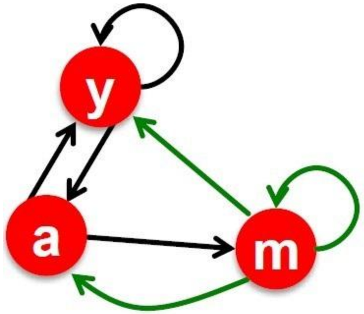
- Theory of Markov chains
- Fact: For any start vector, the power method applied to a Markov transition matrix **P** will converge to a unique positive stationary vector as long as **P** is stochastic, irreducible and aperiodic.

# Make M Stochastic

- Stochastic: Every column sums to 1
- A possible solution: add green links

$$A = M + a^T \left(\frac{1}{n} e\right)$$

- $a_i = 1$ if node $i$ has out degree 0, otherwise 0.
- $e \ldots$ vector of all 1s

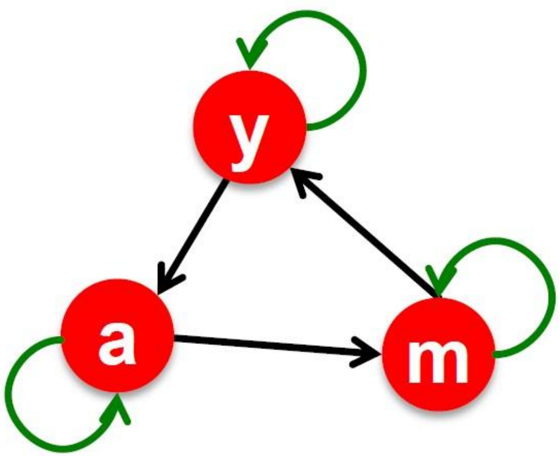



| | y | a | m |
|---|---|---|---|
| y | ½ | ½ | 1/3 |
| a | ½ | 0 | 1/3 |
| m | 0 | ½ | 1/3 |

$r_y = r_y/2 + r_a/2 + r_m/3$
$r_a = r_y/2 + r_m/3$
$r_m = r_a/2 + r_m/3$

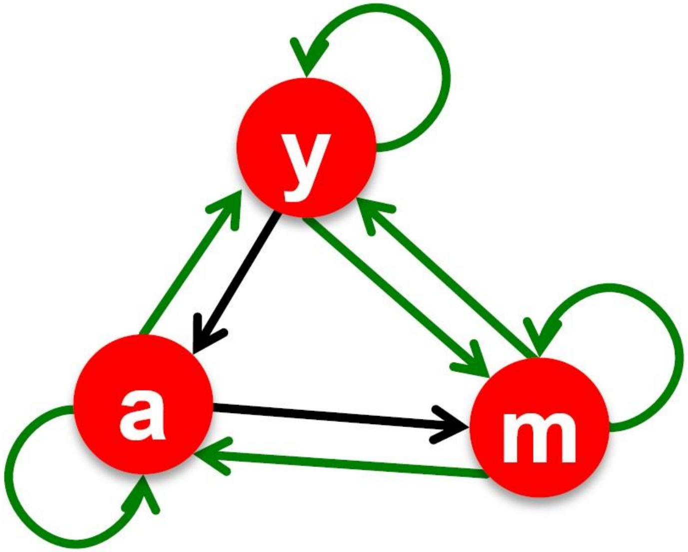
# Make M Aperiodic

- A chain is periodic if there exists **k > 1** such that the interval between two visits to some state **s** is always a multiple of **k**.
- A possible solution: add green links



# Make M Irreducible

- From any state, there is a non-zero probability of going from any one state to any another
- A possible solution: add green links



# Solution: Random Jumps

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

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

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 [Brin-Page,98]

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

- The Google Matrix A:

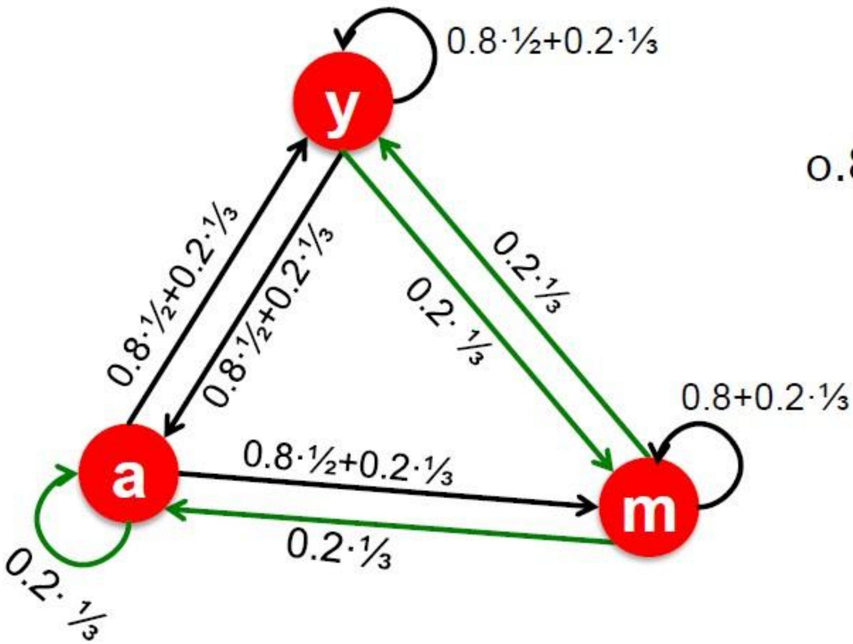
$$A = \beta M + (1-\beta)\frac{1}{n}\mathbf{e} \cdot \mathbf{e}^{\mathbf{T}}$$

- A is stochastic, aperiodic and irreducible, so

$$\mathbf{r}^{(\mathbf{t+1})} = \mathbf{A} \cdot \mathbf{r}^{(\mathbf{t})}$$

- In practice $\beta = 0.8, 0.9$ (make 5 steps and jump)

# Random Teleports ( $\beta = 0.8$ )




$$
0.8 \underbrace{\begin{bmatrix} 1/2 & 1/2 & 0 \\ 1/2 & 0 & 0 \\ 0 & 1/2 & 1 \end{bmatrix}}_{\mathbf{M}} + 0.2 \underbrace{\begin{bmatrix} 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \end{bmatrix}}_{\mathbf{1/n \cdot 1 \cdot 1^T}}
$$

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

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

# In-class Practice

- [Go to Practice]

# Computing PageRank

- Key step is matrix-vector multiplication
  - $r^{new} = A \cdot r^{old}$
- Easy if we have enough main memory to hold $A, r^{old}, r^{new}$
- Say $N$=1 billion pages
  - 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!

# Matrix Formulation

- Suppose there are ***N*** pages
  - Consider page j, with $d_j$ out-links
  - We have $M_{ij} = 1/|d_j|$ when $j \to i$ and $M_{ij} = 0$ otherwise
- The random teleport is equivalent to:
  - Adding a teleport link from j to every other page and setting transition prob. to $(1-\beta)/N$
  - Reducing the prob. of following each out-link from $1/|d_j|$ to $\beta/|d_j|$

# Rearranging the Equation

- $r = A \cdot r$, where $A_{ij} = \beta M_{ij} + \frac{1-\beta}{N}$
- $r_i = \sum_{j=1}^{N} A_{ij} \cdot r_j$
- $$r_i = \sum_{j=1}^{N} \left[\beta M_{ij} + \frac{1-\beta}{N}\right] \cdot r_j$$
  $$= \sum_{j=1}^{N} \beta M_{ij} \cdot r_j + \frac{1-\beta}{N} \sum_{j=1}^{N} r_j$$
  $$= \sum_{j=1}^{N} \beta M_{ij} \cdot r_j + \frac{1-\beta}{N}, \text{ since } \sum r_j = 1$$
- So we get: $r = \beta M \cdot r + \left[\frac{1-\beta}{N}\right]_N$

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

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

# Spare Matrix Formulation

- We just rearranged the PageRank equation

$$r = \beta M \cdot r + \left[\frac{1-\beta}{N}\right]_N$$

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

# PageRank: The Complete Algorithm

- **Input**: Graph **G** and parameter $\beta$
  - Directed graph **G** with **spider traps** and **dead ends**
  - Parameter $\beta$
- **Output**: PageRank vector $r$
  - Set: $r_j^{(0)} = \frac{1}{N}, t = 1$
  - do:
    - $\forall j$: $r'^{(t)}_j = \sum_{i \to j} \beta \frac{r_i^{(t-1)}}{d_i}$
      $r'^{(t)}_j = 0$ if in-deg. of $j$ is 0
    - Now re-insert the leaked PageRank:
      $\forall j : r_j^{(t)} = r'^{(t)}_j + \frac{1-S}{N}$ where $S = \sum_j r'^{(t)}_j$
    - $t = t + 1$
  - while $\sum_j |r_j^{(t)} - r_j^{(t-1)}| > \epsilon$

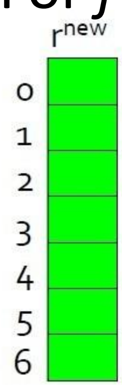
# Sparse Matrix Encoding

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

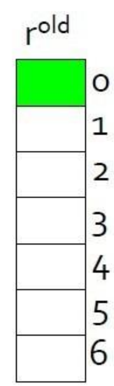
| source node | degree | destination nodes |
|---|---|---|
| 0 | 3 | 1, 5, 7 |
| 1 | 5 | 17, 64, 113, 117, 245 |
| 2 | 2 | 13, 23 |

# Basic Algorithm: Update Step

- Assume enough RAM to fit $r^{new}$ into memory
  - Store $r^{old}$ and matrix *M* on disk
- Then 1 step of power-iteration is:
  - Initialize all entries of $r^{new}$ to $(1-\beta)/N$
  - For each page p (of out-degree n):
    - Read into memory: *p*, *n*, $dest_1$ , …, $dest_n$ , $r^{old}(p)$
    - For *j*=1…*n*: $r^{new}(dest_j)$ += $\beta\, r^{old}(p)/n$



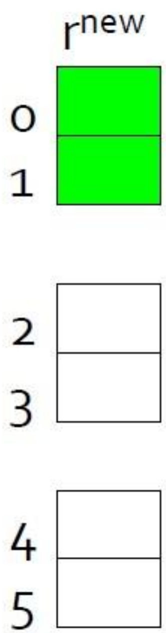

| src | degree | destination |
|---|---|---|
| 0 | 3 | 1, 5, 6 |
| 1 | 4 | 17, 64, 113, 117 |
| 2 | 2 | 13, 23 |



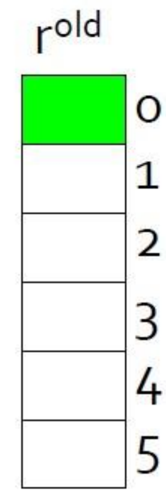
# Analysis

- Assume enough RAM to fit $r^{new}$ into memory
  - Store $r^{old}$ and matrix *M* on disk
- In each iteration, we have to:
  - Read $r^{old}$ and *M*
  - Write $r^{new}$ back to disk
  - IO cost = $2|r| + |M|$
- Question:
  - What if we could not even fit $r^{new}$ in memory

# Block-based Update Algorithm



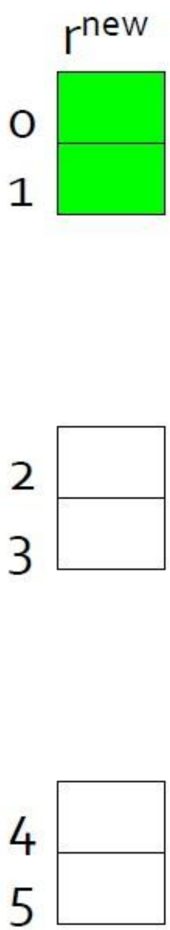

| src | degree | destination |
|---|---|---|
| 0 | 4 | 0, 1, 3, 5 |
| 1 | 2 | 0, 5 |
| 2 | 2 | 3, 4 |



# Analysis of Block Update

- Similar to nested-loop join in databases
  - Break $r^{new}$ into *k* blocks that fit in memory
  - Scan *M* and $r^{old}$ once for each block
- *k* scans of M and $r^{old}$
  - $k(|M|+|r|) + |r| = k|M| + (k+1)|r|$
- Can we do better?
  - Hint: *M* is much bigger than *r* (approx. 10-20x), so we must avoid reading it *k* times per iteration

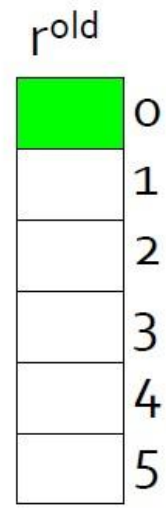
# Block-Strip Update Algorithm




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

| | | |
|---|---|---|
| 0 | 4 | 3 |
| 2 | 2 | 3 |

| | | |
|---|---|---|
| 0 | 4 | 5 |
| 1 | 3 | 5 |
| 2 | 2 | 4 |

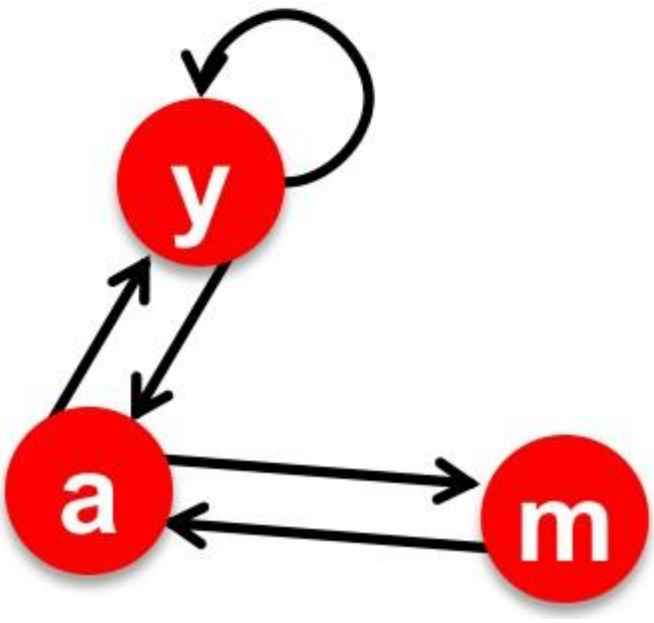


# Block-Strip Analysis

- Break *M* into stripes
  - Each strip contains only destination nodes in the corresponding block of $r^{new}$
- Some additional overhead per stripe
  - But it is usually worth it
- Cost per iteration

$$|M|(1+\epsilon)+(k+1)|r|$$

# In-class Practice




- Compute the final PageRank Score of the given graph.

Go back