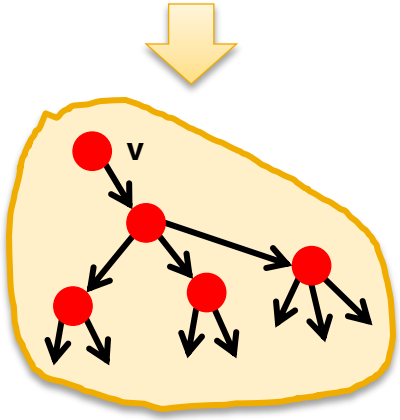
Link Analysis: PageRank

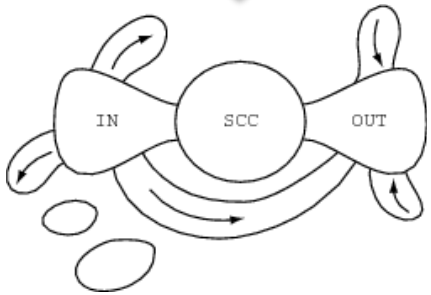
CS224W: Machine Learning with Graphs
Jure Leskovec, Stanford University
http://cs224w.stanford.edu

Web as a Graph

Structure of the Web

- **Today we will talk about how does the Web graph look like:**
  - **1)** We will take a real system: **the Web**
  - **2)** We will represent it as a **directed graph**
  - **3)** We will use the language of graph theory
    - **Strongly Connected Components**
  - **4)** We will design a **computational experiment:**
    - Find In- and Out-components of a given node ***v***
  - **5) We will learn something about the structure of the Web: BOWTIE!**

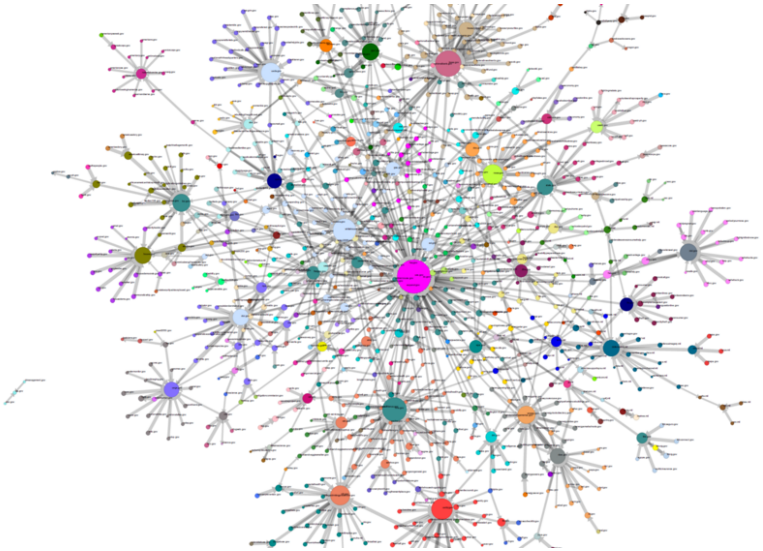




The Web as a Graph

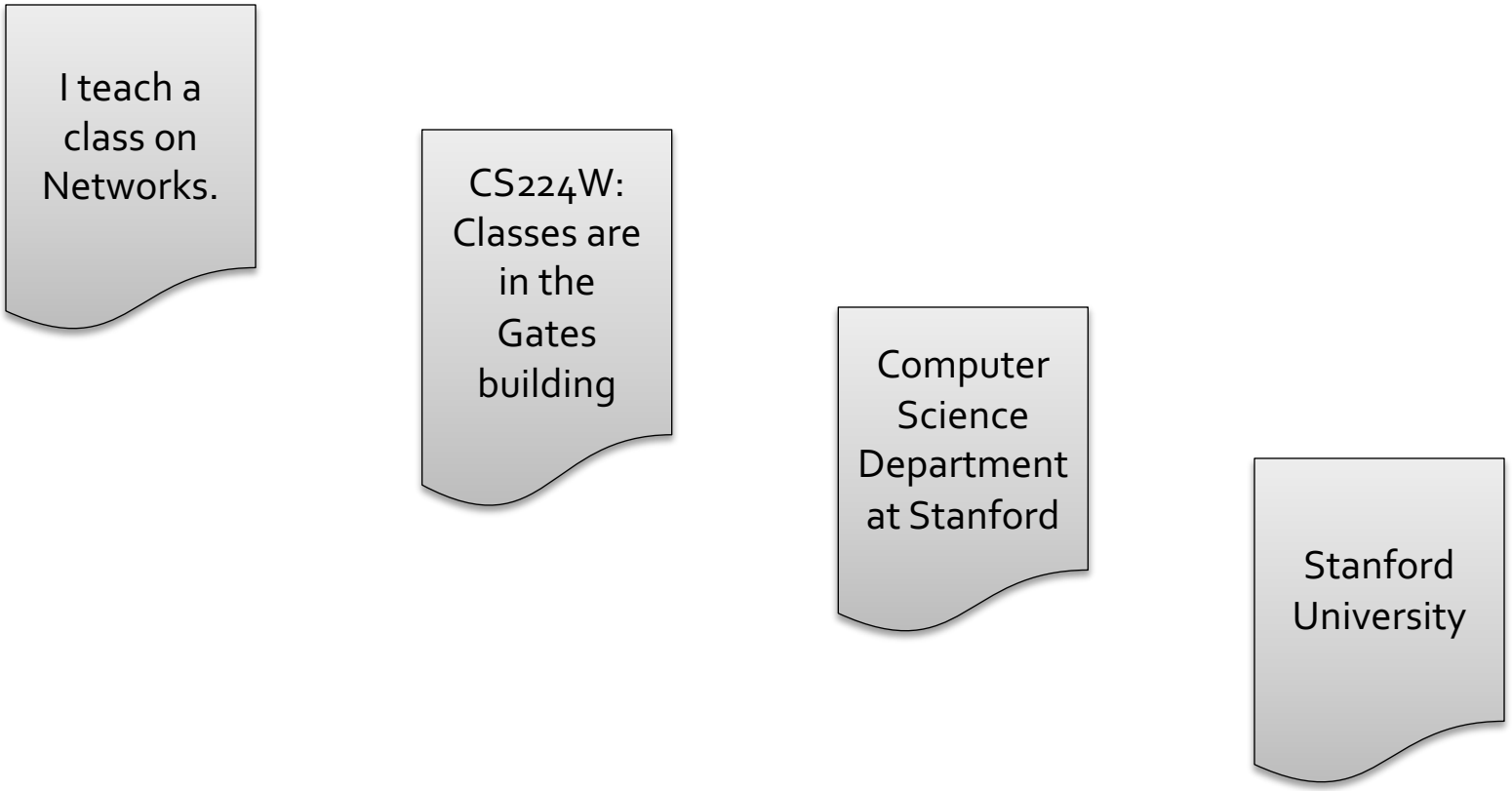
Q: What does the Web "look like" at a global level?

- **Web as a graph:**
  - Nodes = web pages
  - Edges = hyperlinks

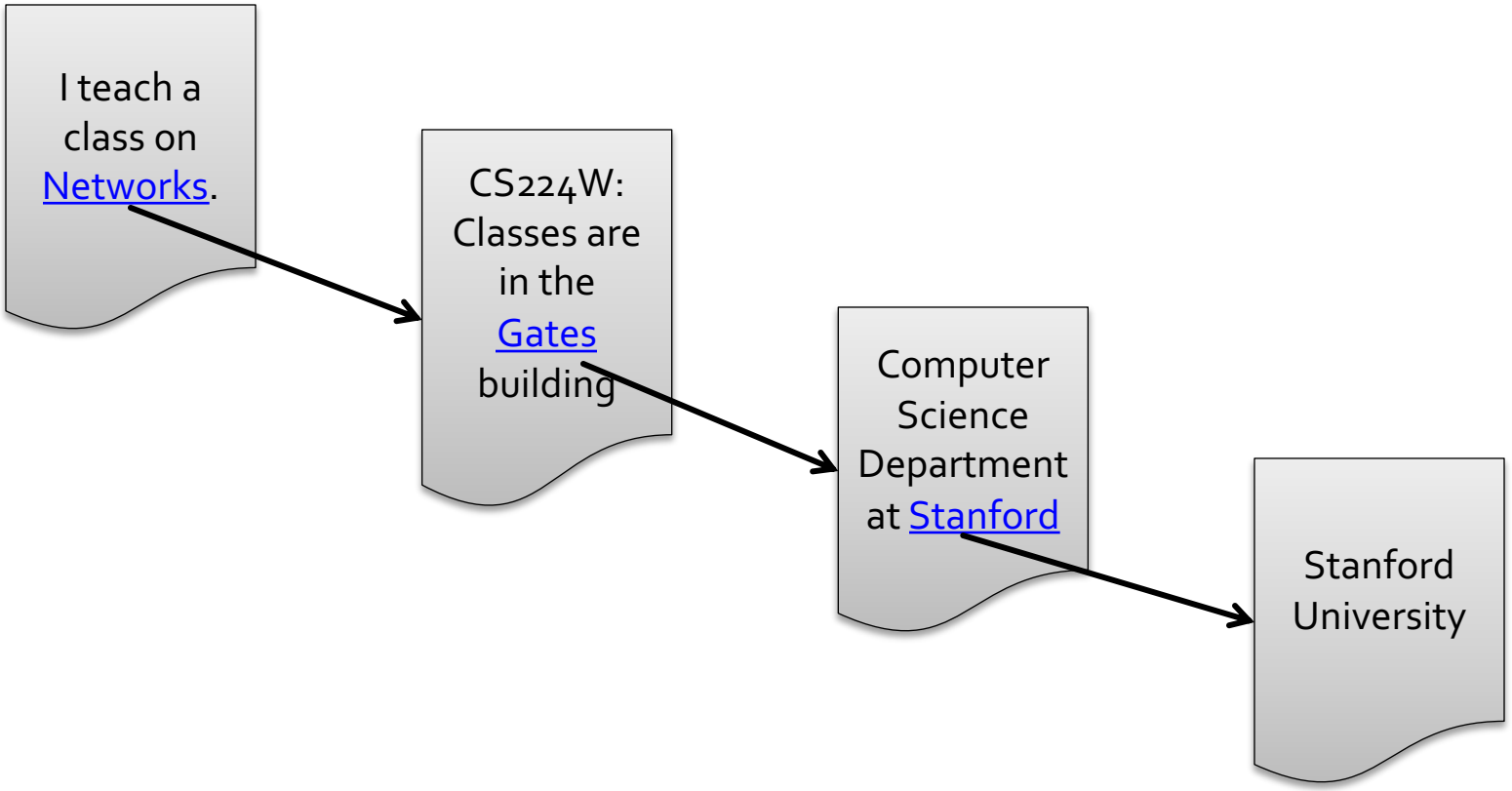


  - **Side issue:** What is a node?
    - Dynamic pages created on the fly
    - "dark matter" – inaccessible database generated pages

The Web as a Graph

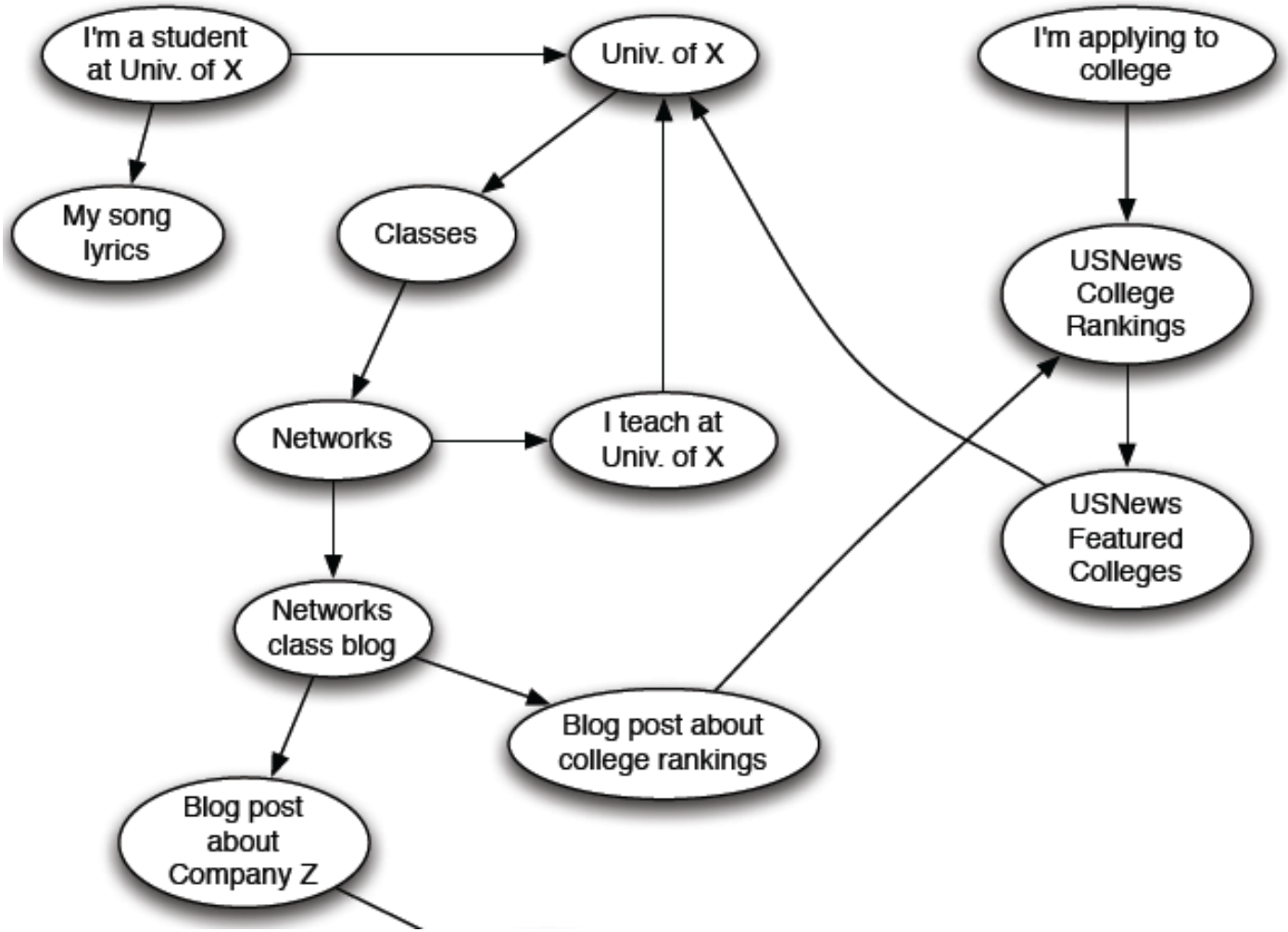


The Web as a Graph

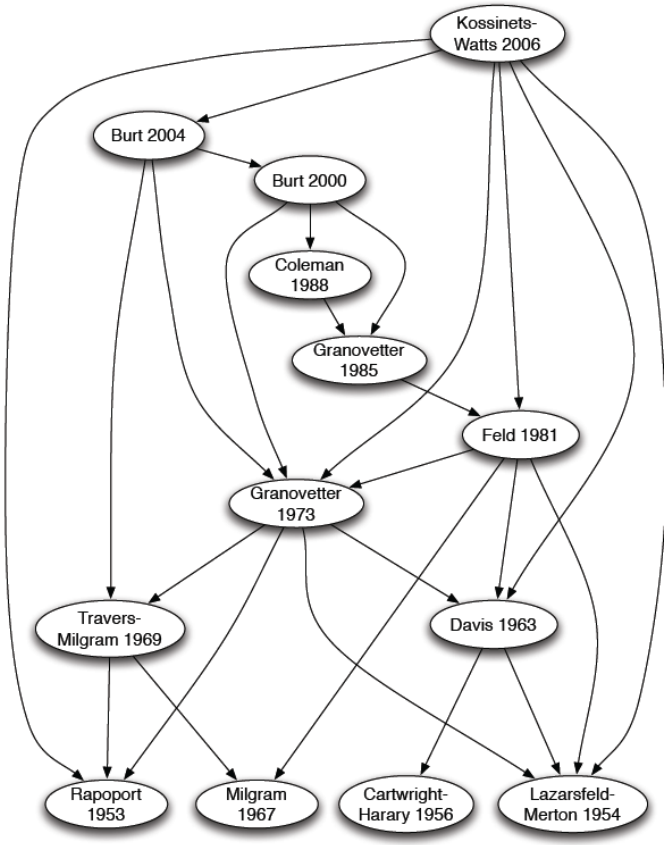


- In early days of the Web links were **navigational**
- Today many links are **transactional** (used not to navigate from page to page, but to post, comment, like, buy, …)

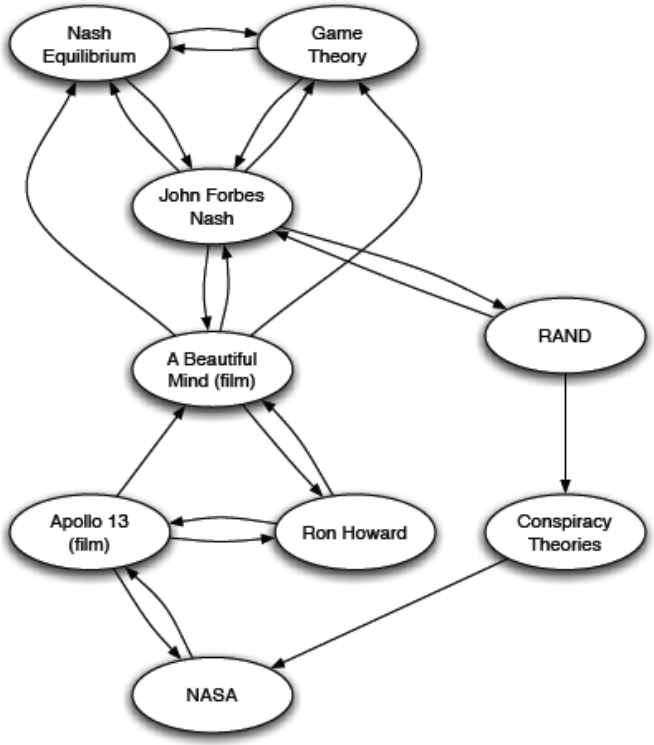
The Web as a Directed Graph



Other Information Networks



Citations



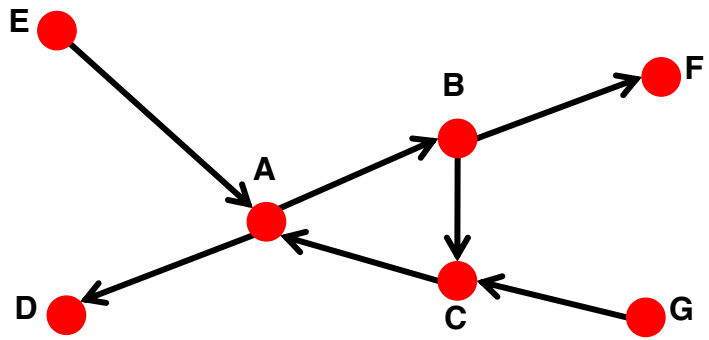
References in an Encyclopedia

What Does the Web Look Like?

- How is the Web linked?
- What is the "map" of the Web?

Web as a directed graph [Broder et al. 2000]:

- Given node v, what nodes can v reach?
- What other nodes can reach v?



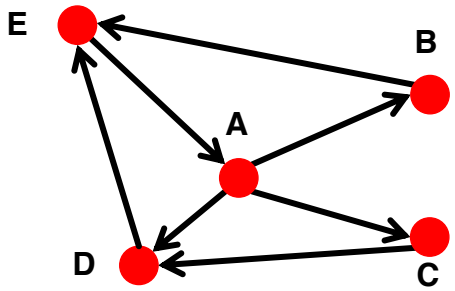
$$In(v) = \{w \mid w \text{ can reach } v\}$$
$$Out(v) = \{w \mid v \text{ can reach } w\}$$

For example:
In(A) = {A,B,C,E,G}
Out(A)={A,B,C,D,F}

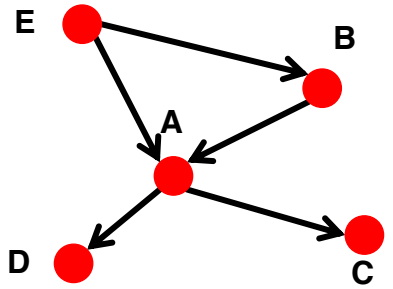
Reasoning about Directed Graphs

- **Two types of directed graphs:**
  - **Strongly connected:**
    - Any node can reach any node via a directed path

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



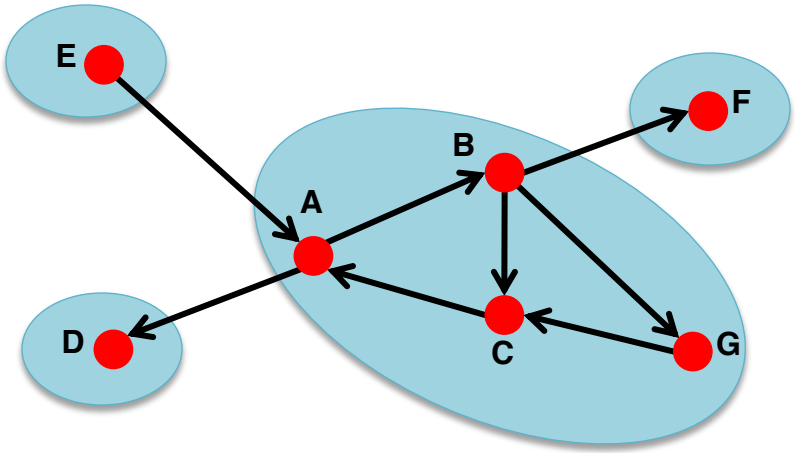
  - **Directed Acyclic Graph (DAG):**
    - Has no cycles: if u can reach v, then v cannot reach u



- **Any directed graph (the Web) can be expressed in terms of these two types!**
  - Is the Web a big strongly connected graph or a DAG?

Strongly Connected Component

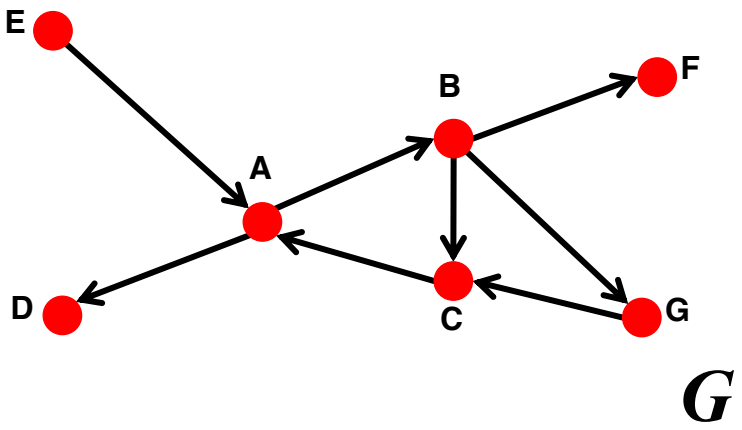
- **A Strongly Connected Component (SCC)** is a set of nodes S so that:
  - Every pair of nodes in S can reach each other
  - There is no larger set containing S with this property



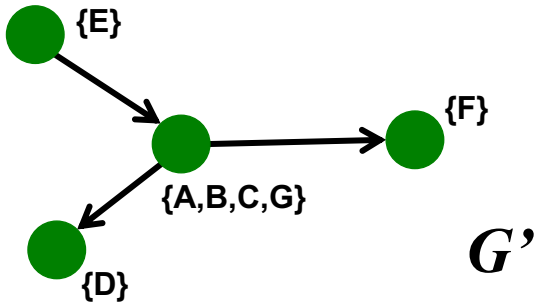
Strongly connected components of the graph: {A,B,C,G}, {D}, {E}, {F}

Strongly Connected Component

- **Fact: Every directed graph is a DAG on its SCCs**
  - **(1)** SCCs partitions the nodes of G
    - That is, 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



(1) Strongly connected components of graph G: {A,B,C,G}, {D}, {E}, {F}
(2) G' is a DAG:



Structure of the Web

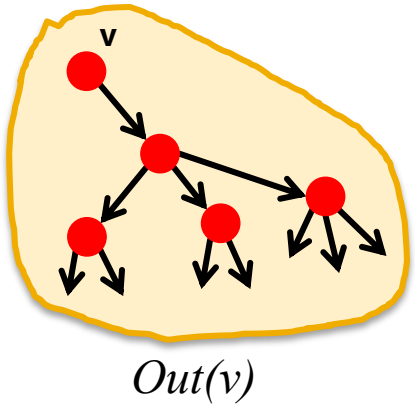
- **Broder et al.: Altavista web crawl (Oct '99)**
  - Web crawl is based on a large set of starting points accumulated over time from various sources, including voluntary submissions.
  - 203 million URLS and 1.5 billion links

Goal: Take a large snapshot of the Web and try to understand how its SCCs "fit together" as a DAG

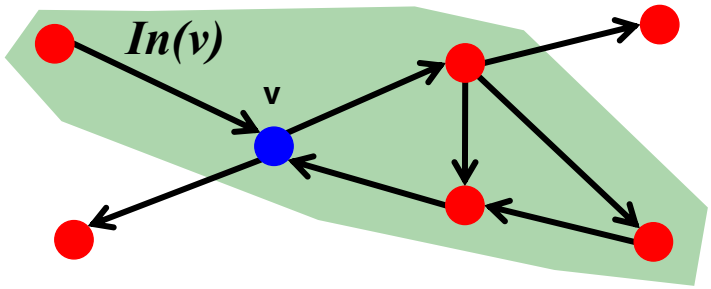
Tomkins, Broder, and Kumar

Graph Structure of the Web

- **Computational issue:**
  - Want to find a SCC containing node v?

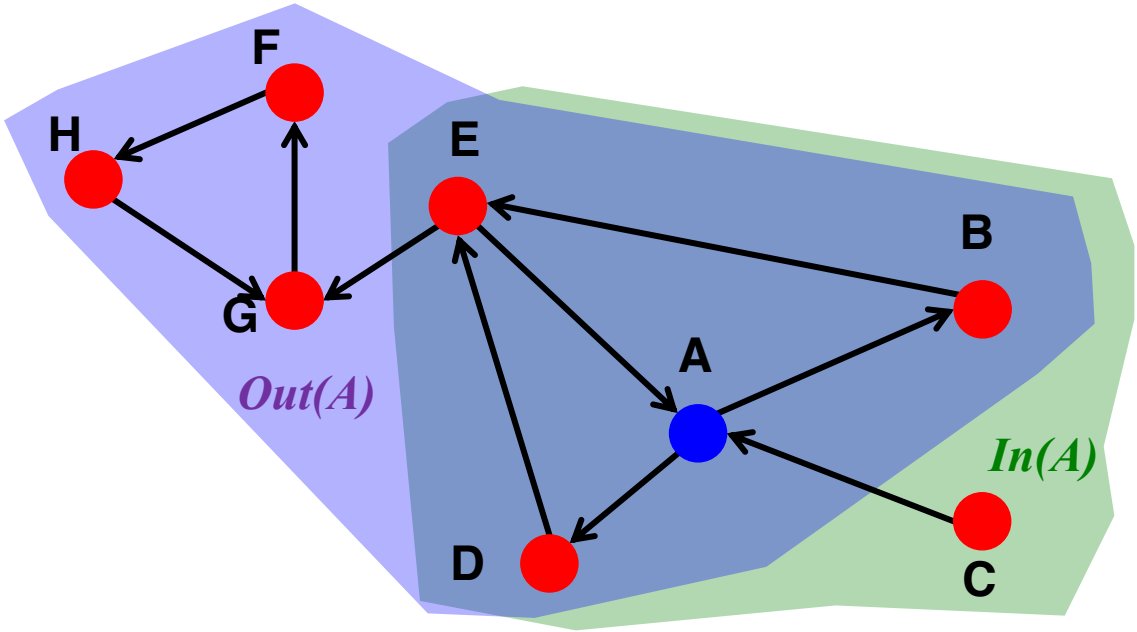


- **Observation:**
  - ***Out(v)*** … nodes that can be reached from v
  - **SCC containing v is:** $Out(v) \cap In(v)$
    $= Out(v,G) \cap Out(v,G')$, where G' is G with all edge directions flipped



Out(A) ∩ In(A) = SCC

- **Example:**



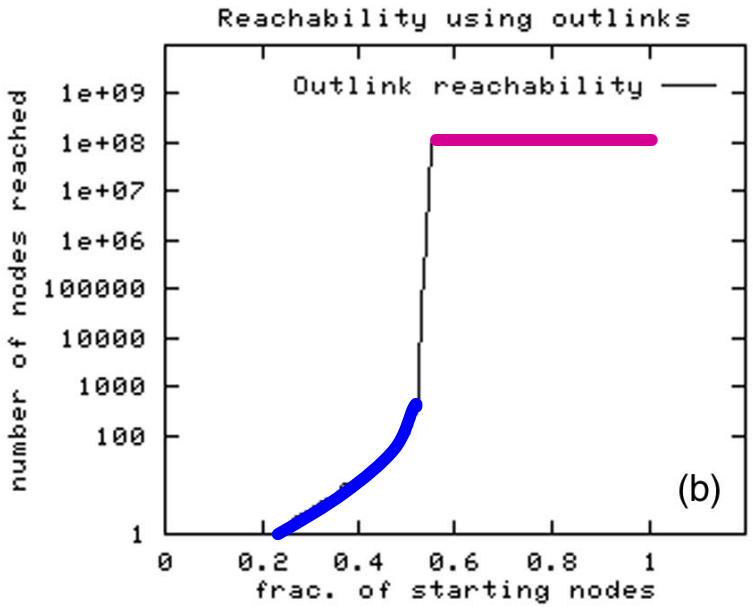
  - Out(A) = {A, B, D, E, F, G, H}
  - In(A) = {A, B, C, D, E}
  - So, SCC(A) = Out(A) ∩ In(A) = {A, B, D, E}

Structure of the Web

- **Directed version of the Web graph:**
  - Altavista crawl from October 1999
    - 203 million URLs, 1.5 billion links

Computation:

- Compute $IN(v)$ and $\mathrm{OUT}(v)$ by starting at random nodes.
- **Observation:** The BFS either visits **many nodes** or **very few**

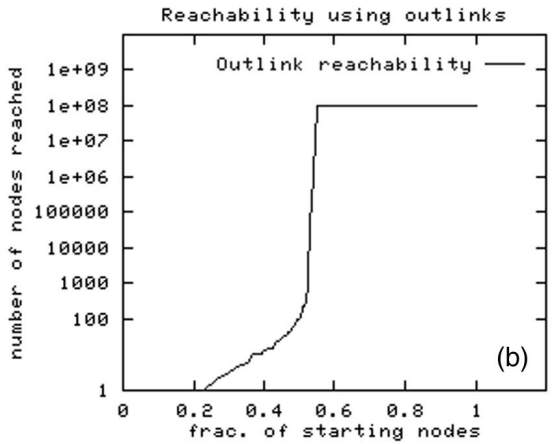


x-axis: rank
y-axis: number of reached nodes

Structure of the Web

Result: Based on IN and OUT of a random node *v*:

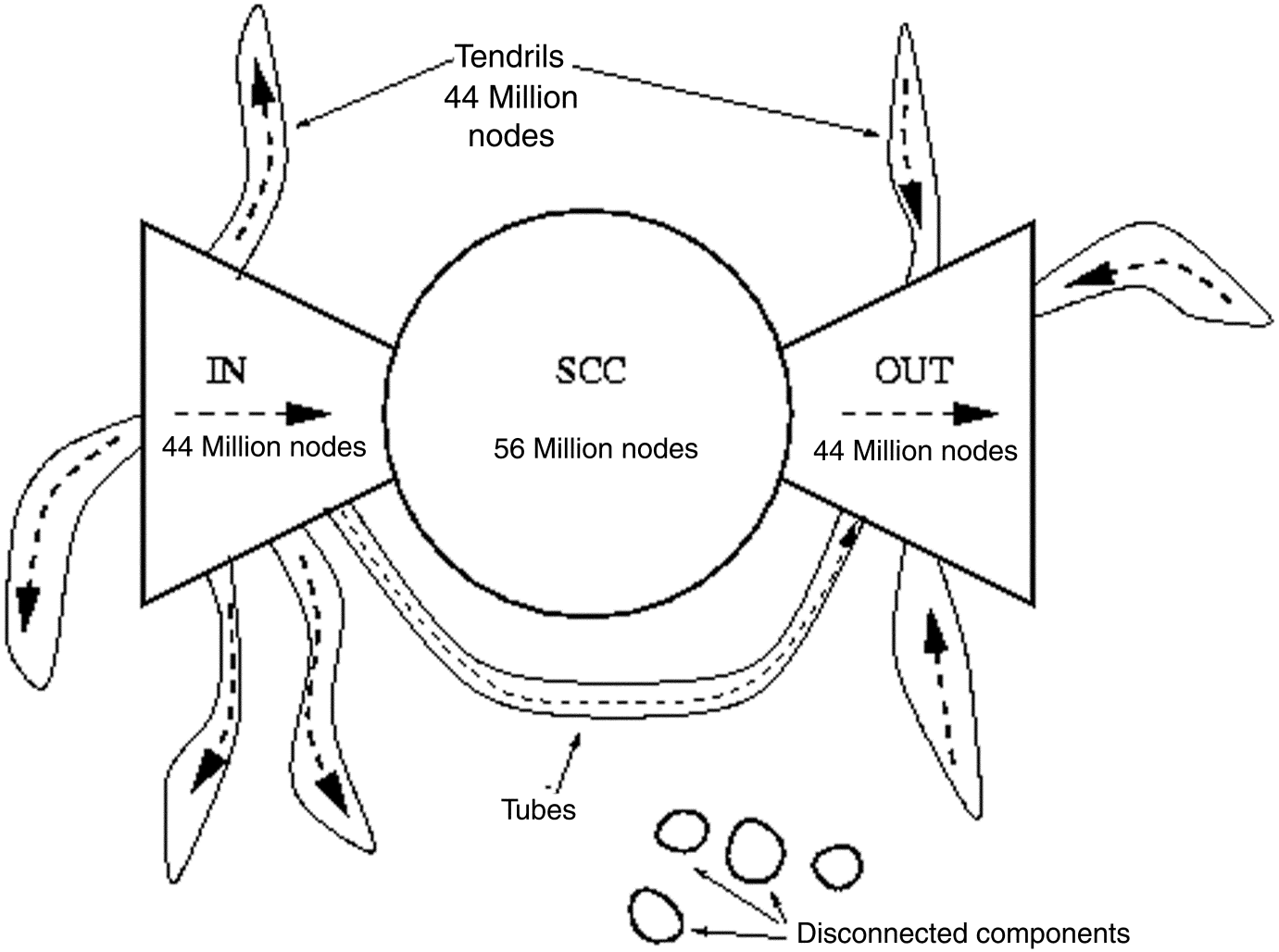
- **Out(v)** ≈ 100 million (**50%** nodes)
- **In(v)** ≈ 100 million (**50%** nodes)
- **Largest SCC:** 56 million (**28%** nodes)



x-axis: rank
y-axis: number of reached nodes

- **What does this tell us about the conceptual picture of the Web graph?**

Bowtie Structure of the Web



203 million pages, 1.5 billion links [Broder et al. 2000]

How to Organize the Web

PageRank

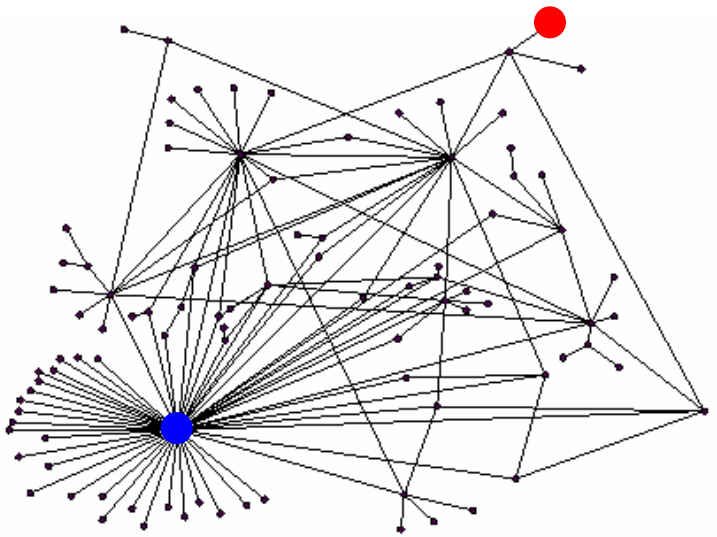
(aka the Google Algorithm)

Ranking Nodes on the Graph

- **All web pages are not equally "important"**

  thispersondoesnotexist.com vs. www.stanford.edu

- There is large diversity in the web-graph node connectivity.
- **So, let's rank the pages using the web graph link structure!**



Link Analysis Algorithms

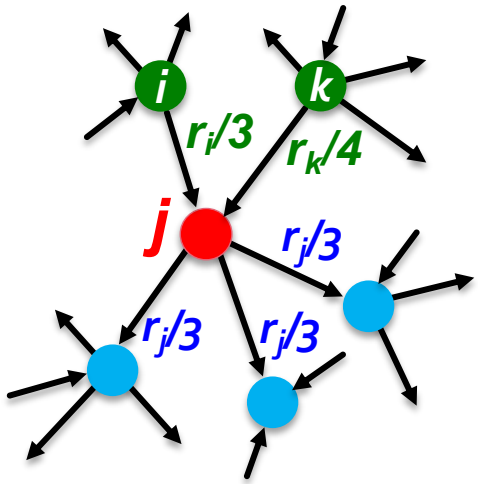
- **We will cover the following Link Analysis approaches** to compute the **importance** of nodes in a graph:
  - PageRank
  - Personalized PageRank
  - Random Walk with Restarts

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.stanford.edu has 23,400 in-links
  - thispersondoesnotexist.com has 1 in-link
- **Are all in-links equal?**
  - Links from important pages count more
  - Recursive question!

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



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

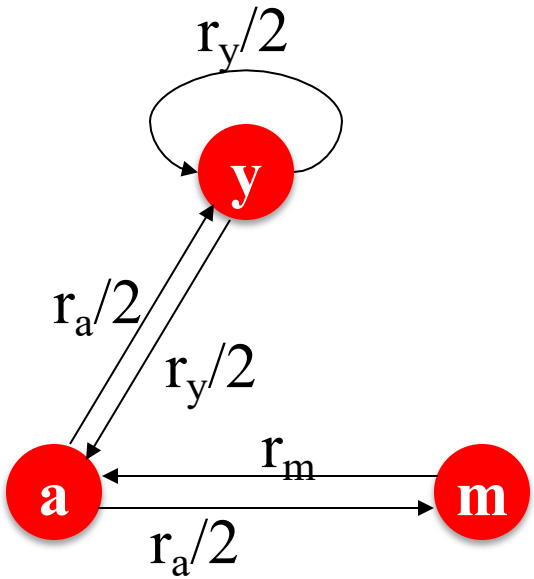
PageRank: The "Flow" Model

- **A page is important if it is pointed to by other important pages**
- **Define a "rank" r_j for node j**

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

d_i … out-degree of node i

The web in 1839



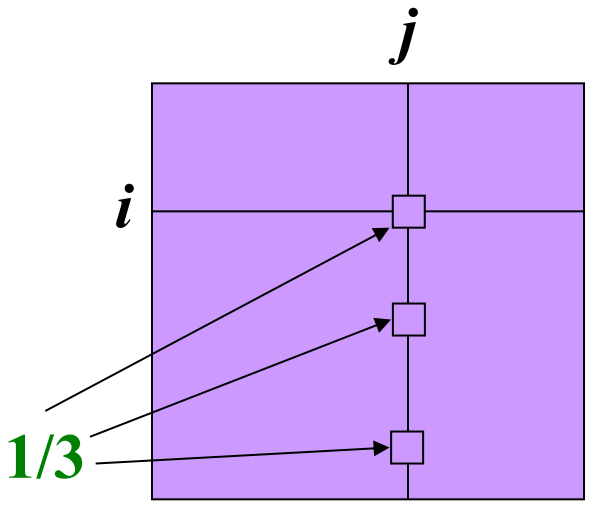
"Flow" equations:

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

You might wonder: Let's just use Gaussian elimination to solve this system of linear equations. Bad idea!

PageRank: Matrix Formulation

- **Stochastic adjacency matrix M**
  - Let page j have d_j out-links
  - If $j \rightarrow i$, then $M_{ij} = \frac{1}{d_j}$
    - M is a **column stochastic matrix**
      - **Columns** sum to **1**



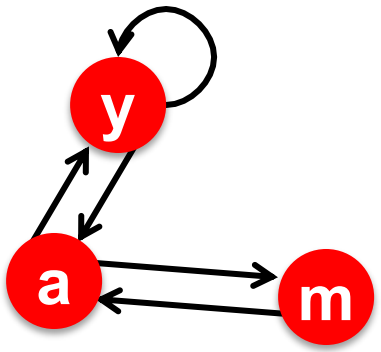
- **Rank vector r:** An entry per page
  - r_i is the importance score of page i
  - $\sum_i r_i = 1$
- **The flow equations can be written**

$$r = M \cdot r$$

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

M

Example: Flow Equations & M



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

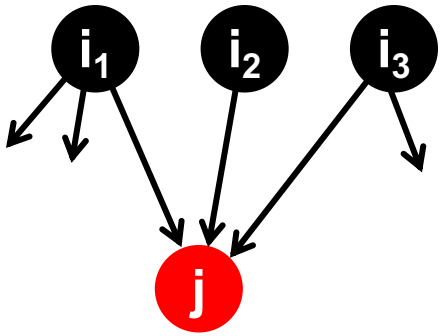
$$r = M \cdot r$$

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

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

Random Walk Interpretation

- **Imagine a random web surfer:**
  - At any time 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



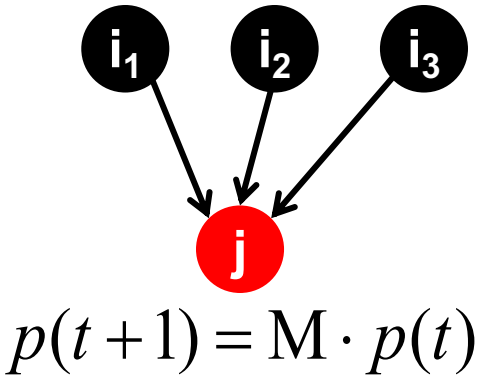
$$r_j = \sum_{i \to j} \frac{r_i}{\mathrm{d}_{\mathrm{out}}(\mathrm{i})}$$

- **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 $t+1$?**
  - Follow a link uniformly at random

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



- Suppose the random walk reaches a 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**

Eigenvector Formulation

- **The flow equations can be written**

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

- So the **rank vector** ***r*** is an **eigenvector** of the stochastic web matrix ***M***
  - Starting from any vector $\boldsymbol{u}$, the limit $\boldsymbol{M}(\boldsymbol{M}(\ldots \boldsymbol{M}(\boldsymbol{M}\,\boldsymbol{u})))$ is the long-term distribution of the surfers.
    - **The math:** Limiting distribution = principal eigenvector of M = PageRank.
      - Note: If $\boldsymbol{r}$ is the limit of $\boldsymbol{MM} \ldots \boldsymbol{Mu}$, then $\boldsymbol{r}$ satisfies the equation $\boldsymbol{r} = \boldsymbol{Mr}$, so r is an eigenvector of $\boldsymbol{M}$ with eigenvalue 1

NOTE: ***x*** is an eigenvector with the corresponding eigenvalue **λ** if:

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

- **We can now efficiently solve for *r*!**

  The method is called Power iteration

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
  - Initialize: $\mathbf{r}^{(0)} = [1/N,\ldots.,1/N]^T$
  - Iterate: $\mathbf{r}^{(t+1)} = \mathbf{M} \cdot \mathbf{r}^{(t)}$
  - Stop when $|\mathbf{r}^{(t+1)} - \mathbf{r}^{(t)}|_1 < \varepsilon$

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

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

About 50 iterations is sufficient to estimate the limiting solution.

PageRank: How to solve?

PageRank: How to solve?

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

- Assign each node an initial page rank
- Repeat until convergence ($\Sigma_i |r_i^{(t+1)} - r_i^{(t)}| < \varepsilon$)
  - Calculate the page rank of each node

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

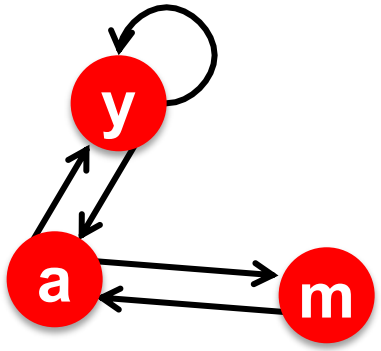
d_i …. out-degree of node i

PageRank: How to solve?

- **Power Iteration:**
  - Set $r_j \leftarrow 1/N$
  - **1:** $r'_j \leftarrow \sum_{i \to j} \frac{r_i}{d_i}$
  - **2:** If $|r - r'| > \varepsilon$:
    - $r \leftarrow r'$
  - **3:** goto **1**
- **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, …



	y	a	m
y	½	½	0
a	½	0	1
m	0	½	0

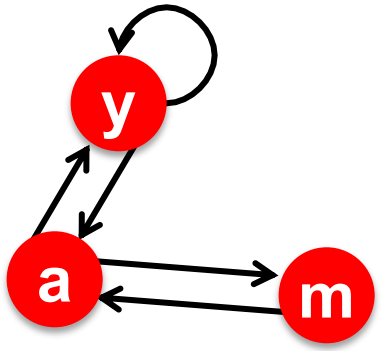
$r_y = r_y/2 + r_a/2$

$r_a = r_y/2 + r_m$

$r_m = r_a/2$

PageRank: How to solve?

- **Power Iteration:**
  - Set $r_j \leftarrow 1/N$
  - **1:** $r'_j \leftarrow \sum_{i \to j} \frac{r_i}{d_i}$
  - **2:** If $|r - r'| > \varepsilon$:
    - $r \leftarrow r'$
  - **3:** goto **1**
- **Example:**



	y	a	m
y	½	½	0
a	½	0	1
m	0	½	0

$r_y = r_y/2 + r_a/2$

$r_a = r_y/2 + r_m$

$r_m = r_a/2$

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

Iteration 0, 1, 2, …

PageRank: Three Questions

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

or equivalently

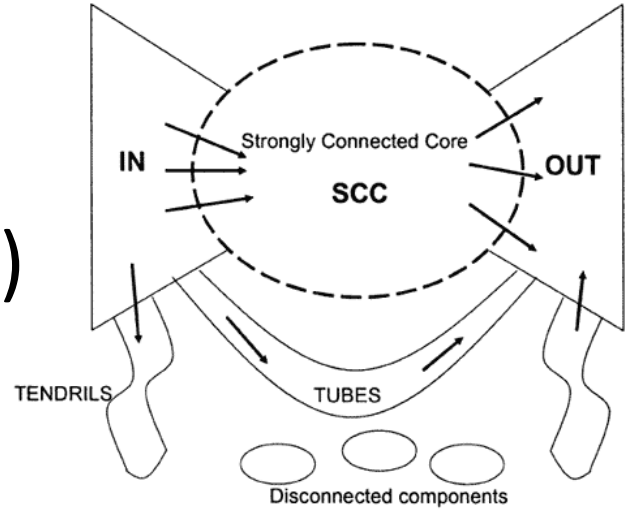
$$r = Mr$$

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

PageRank: Problems

Two problems:

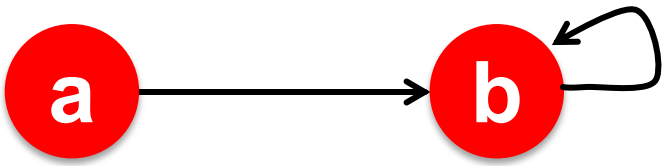
- **(1)** Some pages are **dead ends** (have no out-links)
  - Such pages cause importance to "leak out"



- **(2) Spider traps** (all out-links are within the group)
  - Eventually spider traps absorb all importance

Does this converge?

- **The "Spider trap" problem:**



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

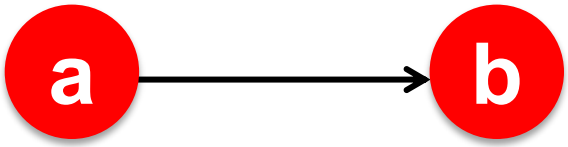
- **Example:**

Iteration: 0, 1, 2, 3…

		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:**



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

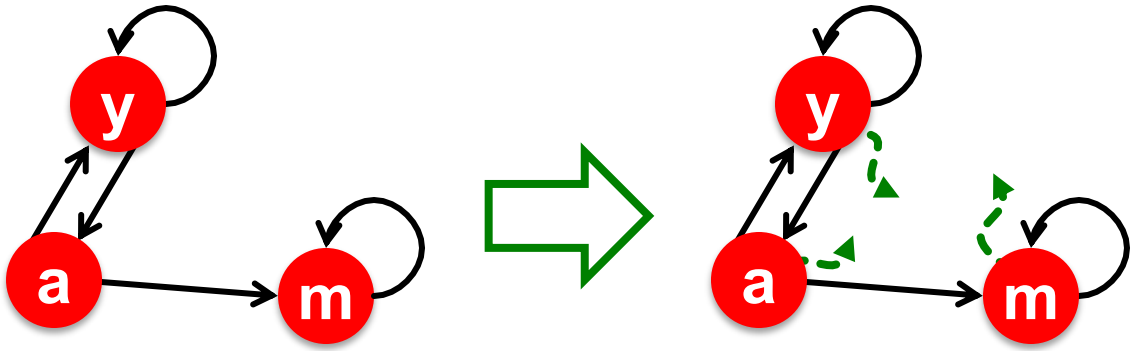
- **Example:**

Iteration: 0, 1, 2, 3…

$$\begin{matrix} r_a \\ r_b \end{matrix} = \left[\begin{array}{c|c|c|c} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{array}\right]$$

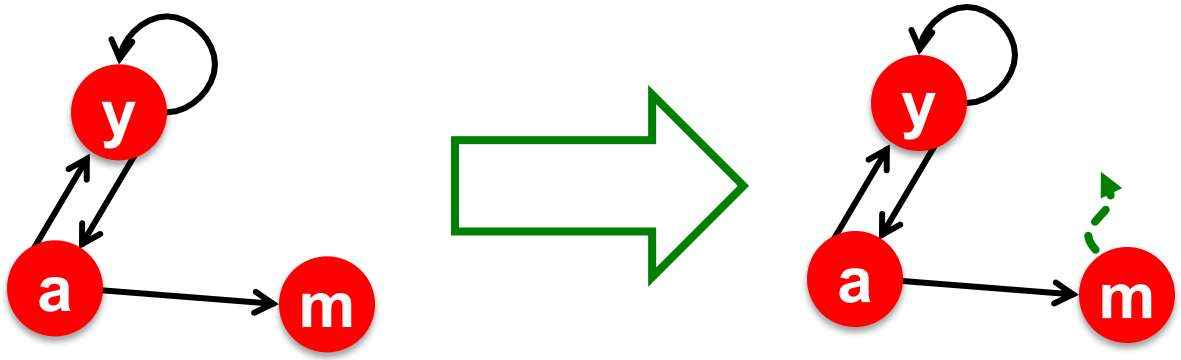
Solution to Spider Traps

- **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 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

- **Teleports:** 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

- **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 $\boldsymbol{M}$ has no dead ends. We can either preprocess matrix $\boldsymbol{M}$ 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*:**

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

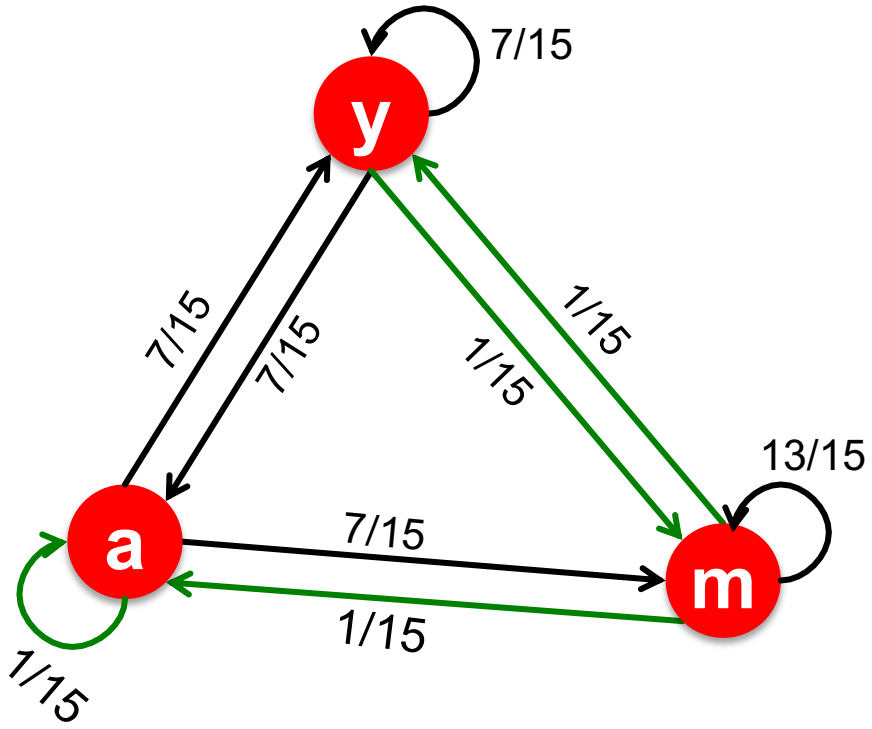
$$A = \beta\, M + (1 - \beta) \left[\frac{1}{N}\right]_{N \times N}$$

- **We have a recursive problem: $\boldsymbol{r = A \cdot r}$**

  And the Power method still works!

- **What is β?**
  - In practice $\beta = 0.8, 0.9$ (make 5 steps on avg., jump)

Random Teleports ($\beta = 0.8$)



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

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

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

A

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

How do we actually compute the PageRank?

Computing PageRank

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

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

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

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

Sparse Matrix Formulation

- We can rearrange the **PageRank equation**

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

  - where $\mathbf{[(1-\beta)/N]_N}$ is a vector with all $\boldsymbol{N}$ entries $\mathbf{(1-\beta)/N}$

- $\boldsymbol{M}$ 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 r^{new} so that it sums to 1**

PageRank: The Complete Algorithm

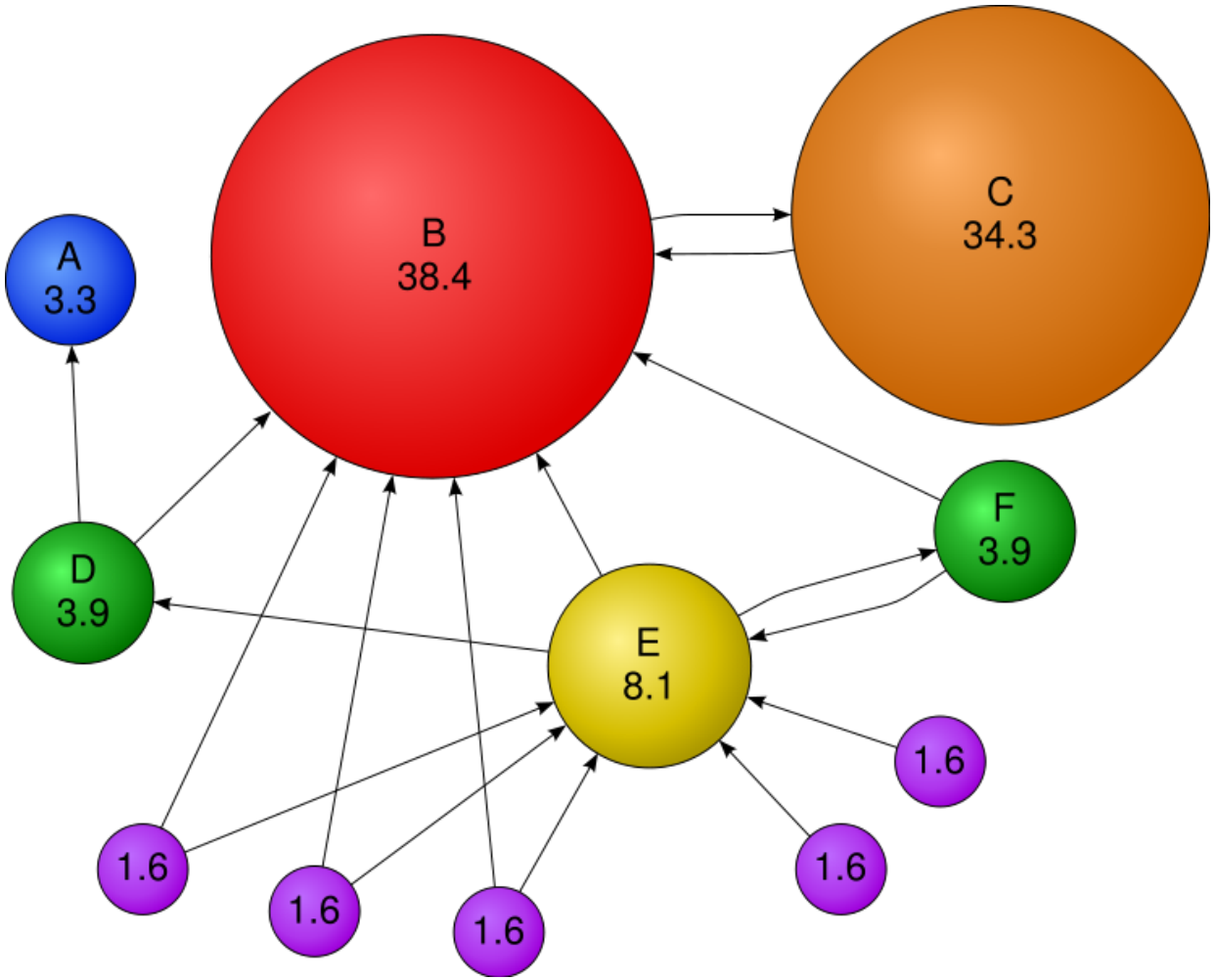
- **Input: Graph G and parameter β**
  - Directed graph G (can have **spider traps** and **dead ends**)
  - Parameter β
- **Output: PageRank vector r^{new}**
  - **Set:** $r_j^{old} = \frac{1}{N}$
  - **repeat until convergence:** $\sum_j \left| r_j^{new} - r_j^{old} \right| < \varepsilon$
    - $\forall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

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

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

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**.

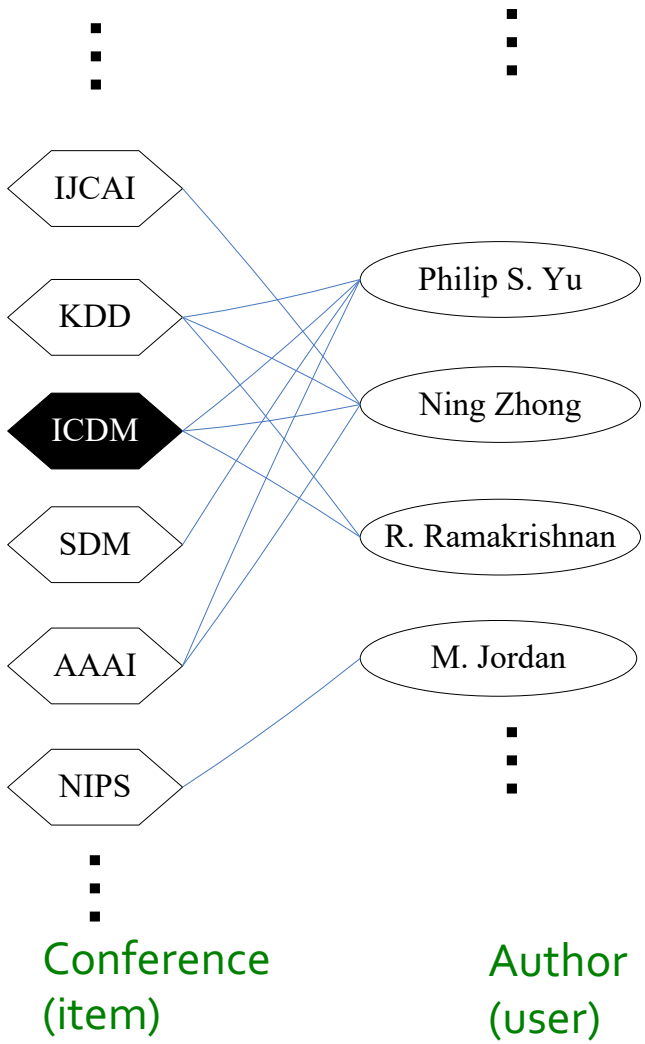
Example



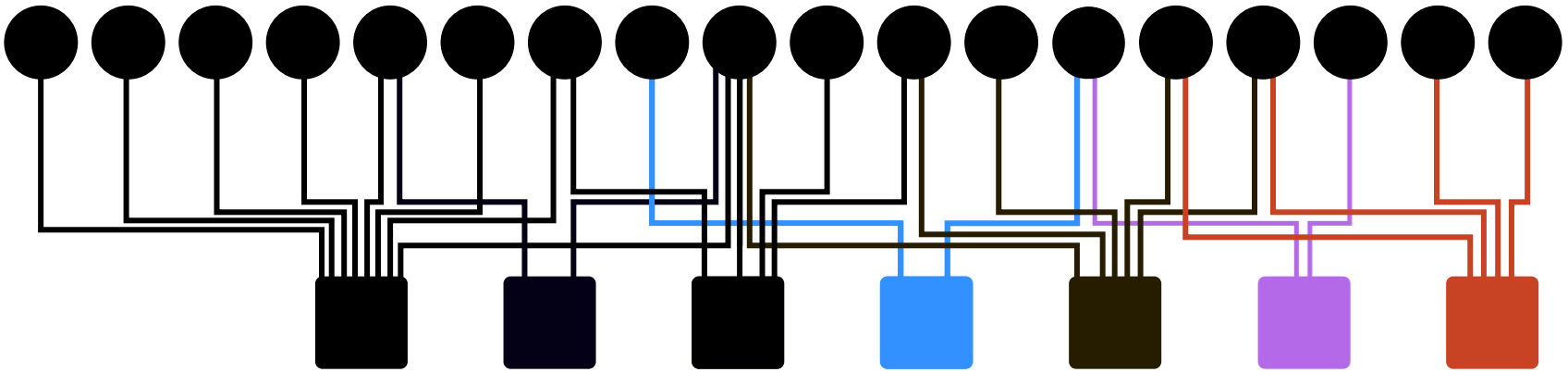
Random Walk with Restarts and Personalized PageRank

Example Application: Graph Search

- **Given:**
  Conferences-to-authors graph
- **Goal:**
  Proximity on graphs
  - **What is most related conference to ICDM?**
  - **What conferences should we recommend to M. Jordan to attend?**

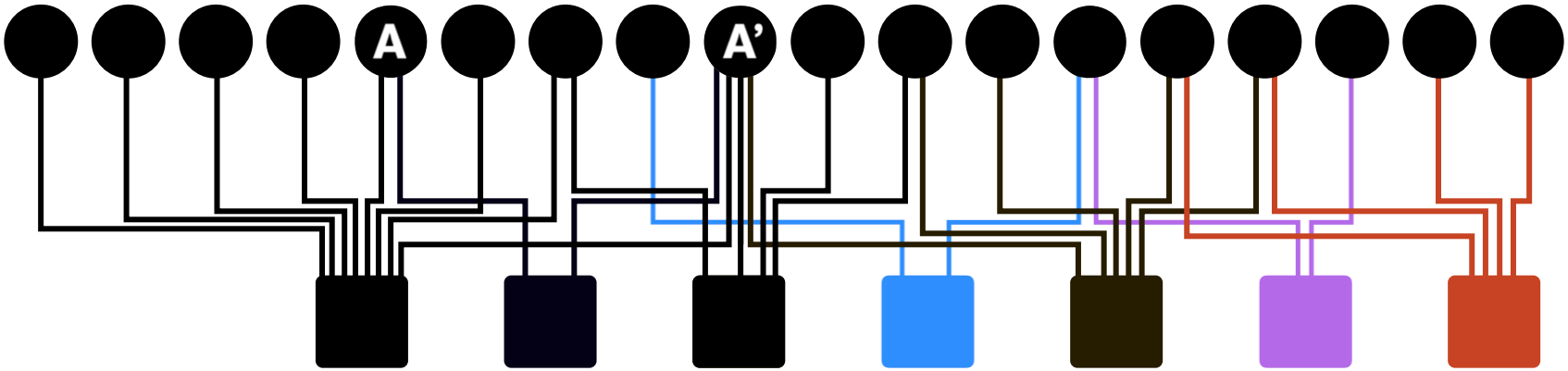


Bipartite User-to-Item Graph



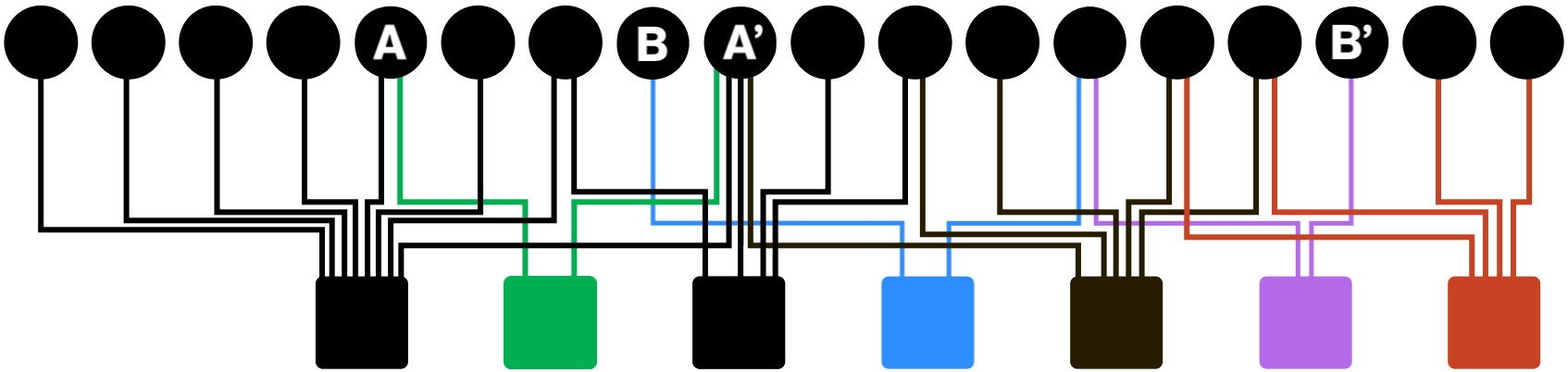
Bipartite User-to-Item Graph

- **Recommender systems:** User to Item graph
  - Collaborative filtering – recommend items based on the users they have in common



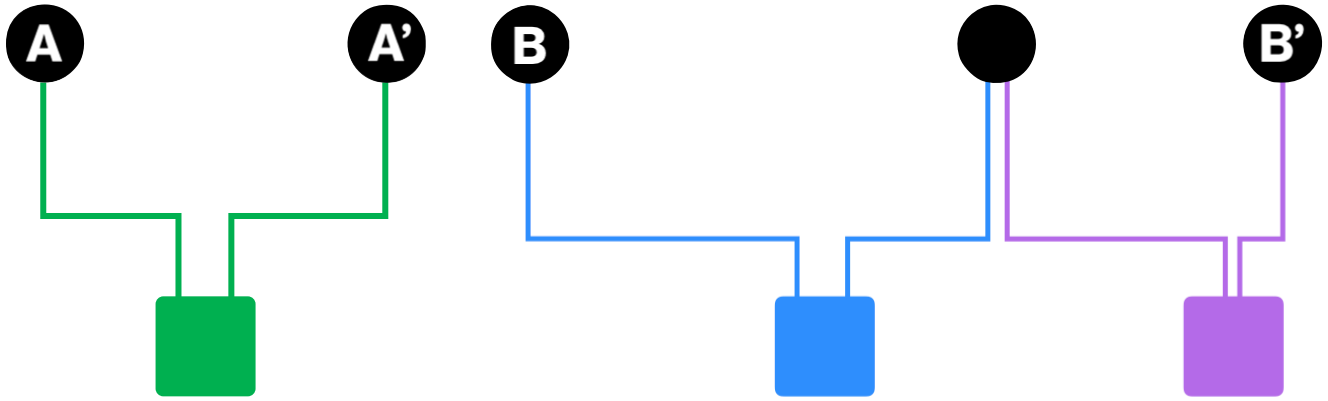
Bipartite User-to-Item Graph

- **Which is more related A,A’ or B,B’?**



Node proximity Measurements

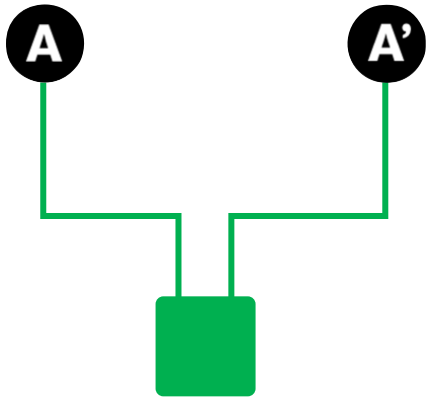
- Which is more related A,A', B,B' or C,C'?



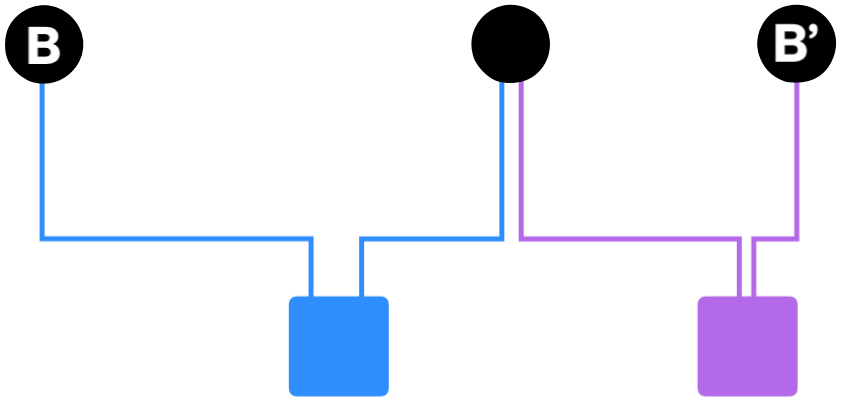
Shortest path

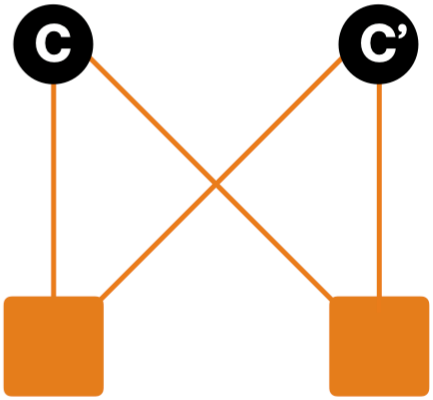
Node proximity Measurements

- Which is more related A,A', B,B' or C,C'?



Shortest path

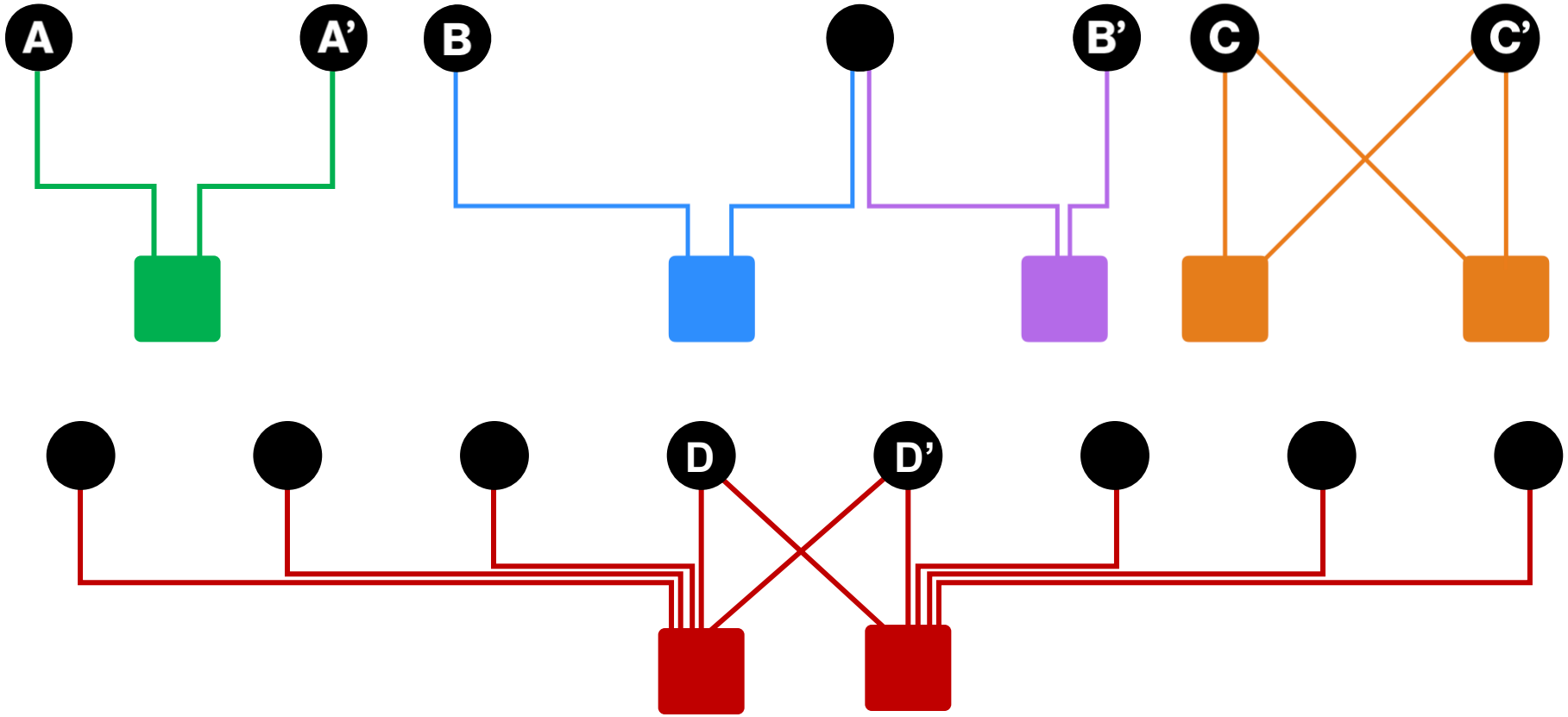




Common Neighbors

Node proximity Measurements

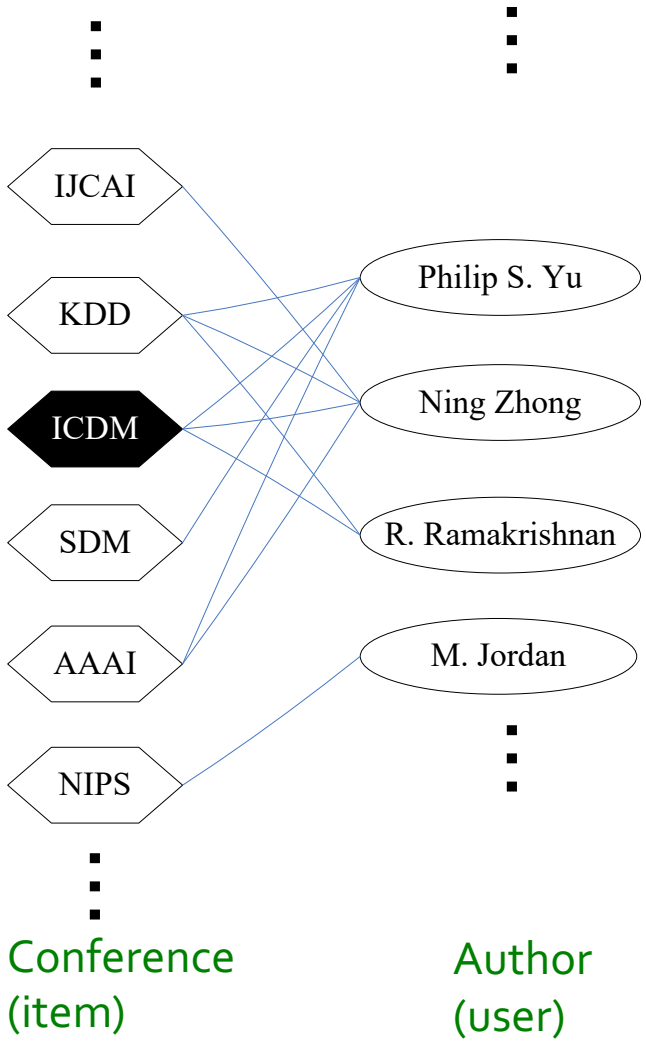
- Which is more related A,A', B,B' or C,C'?



Personalized Page Rank/Random Walk with Restarts

Proximity on Graphs

- **Graphs and web search:**
  - Ranks nodes by "importance"
- **Personalized PageRank:**
  - Ranks proximity of nodes to the teleport nodes S
- **Proximity on graphs:**
  - **Q:** What is most related conference to **ICDM**?
  - **Random Walks with Restarts**
    - Teleport back to the starting node: $S = \{ \text{single node} \}$

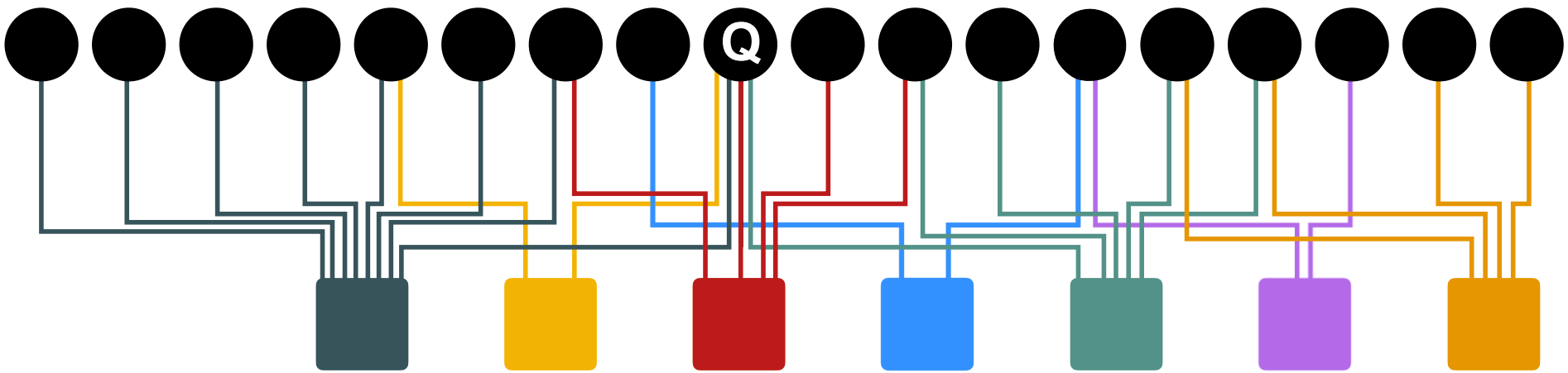


Idea: Random Walks

- **Idea**
  - Every node has some importance
  - Importance gets evenly split among all edges and pushed to the neighbors:
- **Given a set of QUERY_NODES, we simulate a random walk:**
  - Make a step to a random neighbor and record the visit (visit count)
  - With probability ALPHA, restart the walk at one of the QUERY_NODES
  - The nodes with the highest visit count have highest proximity to the QUERY_NODES

Random Walks

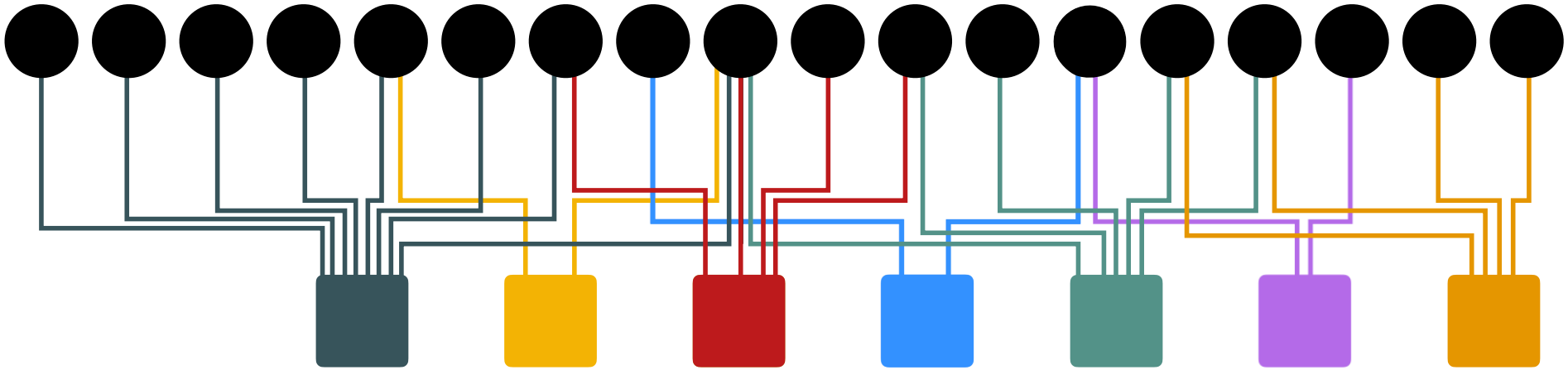
- **Idea:**
  - Every node has some importance
  - Importance gets evenly split among all edges and pushed to the neighbors
- Given a set of QUERY NODES Q, simulate a random walk:



Pixie Random Walk Algorithm

- Proximity to query node(s) ***Q***:

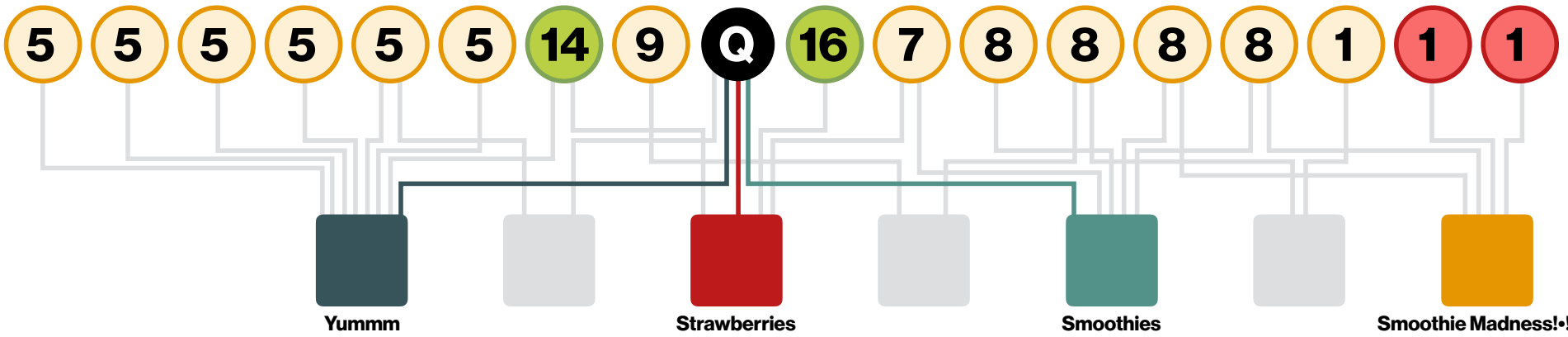
```
ALPHA = 0.5
QUERY_NODES = { Q }
pin_node = QUERY_NODES.sample_by_weight()
for i in range(N_STEPS):
    board_node = pin_node.get_random_neighbor()
    pin_node = board_node.get_random_neighbor()
    pin_node.visit_count += 1
    if random() < ALPHA:
        pin_node = QUERY_NODES.sample_by_weight()
```



Pixie Random Walk Algorithm

- Proximity to query node(s) ***Q***:

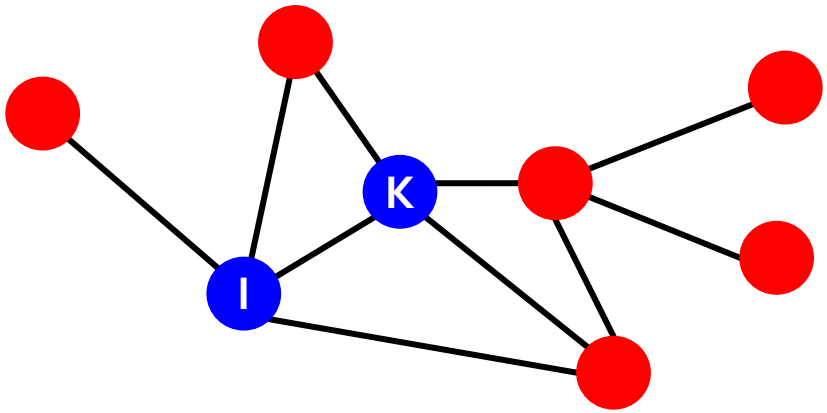
```
ALPHA = 0.5
QUERY_NODES = { ● }
pin_node = QUERY_NODES.sample_by_weight()
for i in range(N_STEPS):
    board_node = pin_node.get_random_neighbor()
    pin_node = board_node.get_random_neighbor()
    pin_node.visit_count += 1
    if random() < ALPHA:
        pin_node = QUERY_NODES.sample_by_weight()
```



Benefits

- **Why is this great?**
- **It considers:**
  - Multiple connections
  - Multiple paths
  - Direct and indirect connections
  - Degree of the node

Random Walks

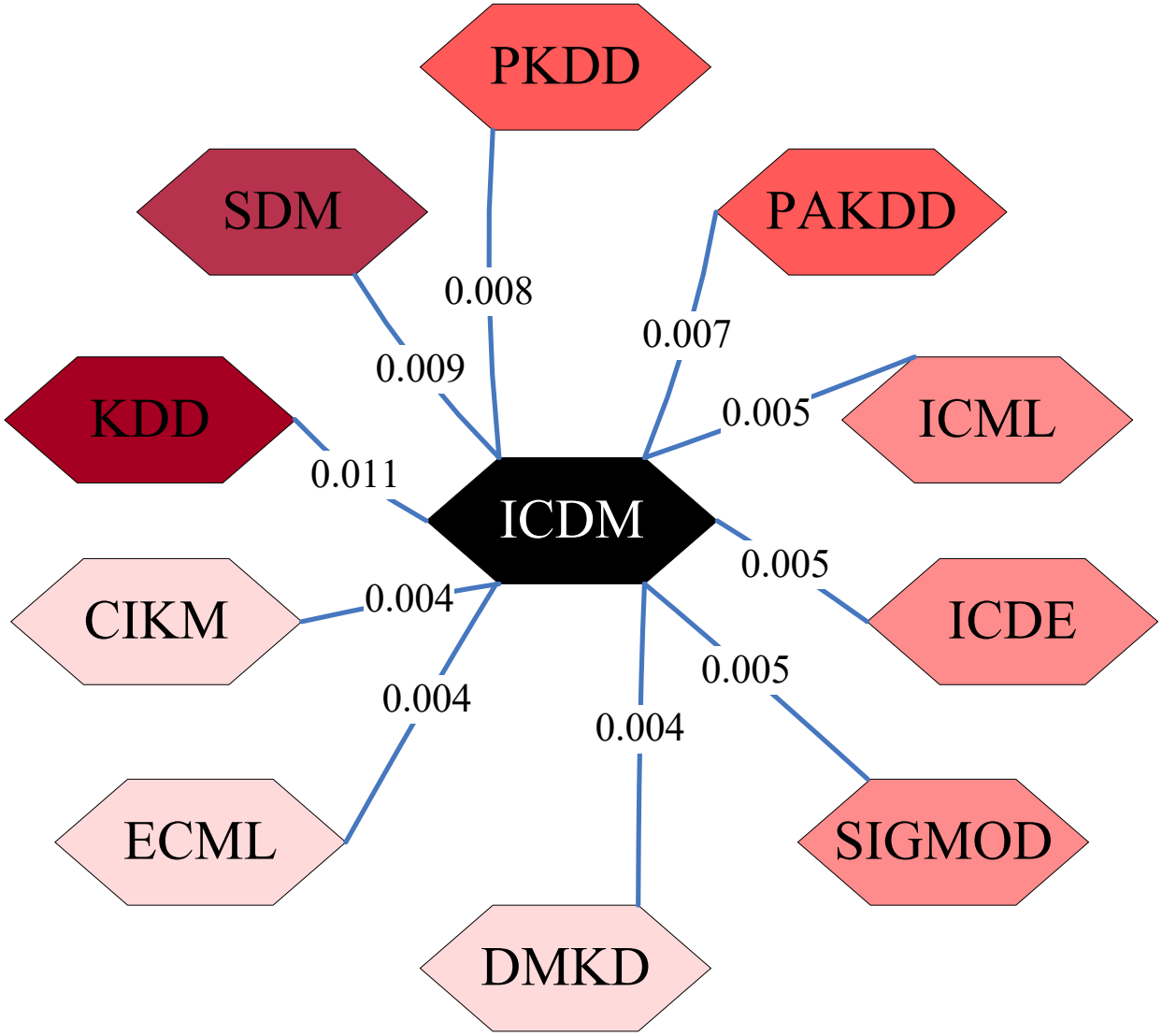


Graph of CS conferences

Q: Which conferences are closest to KDD & ICDM?

A: Personalized PageRank with teleport set S={KDD, ICDM}

Most related conferences to ICDM



PageRank: Summary

- **"Normal" PageRank:**
  - Teleports uniformly at random to any node
  - All nodes have the same probability of surfer landing there: **S =** [0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1]
- **Topic-Specific PageRank** also known as **Personalized PageRank:**
  - Teleports to a topic specific set of pages
  - Nodes can have different probabilities of surfer landing there: **S =** [0.1, 0, 0, 0.2, 0, 0, 0.5, 0, 0, 0.2]
- **Random Walk with Restarts:**
  - Topic-Specific PageRank where teleport is always to the same node. S=[0, 0, 0, 0, **1**, 0, 0, 0, 0, 0, 0]