Information Networks and the WWW

Link Analysis

Overview

- Intro to Web Search
- Crawling
- Web Link Analysis
- Beyond Page Rank
- Link Analysis in other scenarios

Overview

- Intro to Web Search
- Crawling
- Web Link Analysis
- Beyond Page Rank
- Link Analysis in other scenarios

Information Retrieval - Limitations

- The field of information retrieval has dealt with the problem of search for decades before the creation of the Web:
- Automated information retrieval systems starting in the 1960s were designed to search repositories of newspaper articles, scientific papers, patents, legal abstracts, and other document collections in reponse to keyword queries

- Problems with Keyword-based Search
  - Expressing what user wants with just keywords
    - Synonyms (Automobile)
    - Polysemy (Jaguar)

- Problem with Web Search
  - Dynamic nature
    - Can't search for latest information
  - Problem of scarcity to a problem of abundance
    - **Ranking becomes critical**

Web Search Engine - Processes

- Crawling
- Indexing
- Search
- Ranking

History of Web Search

- The first popular search engine on the Web was Yahoo! Search.
  - The first product from Yahoo!, founded by Jerry Yang and David Filo in January 1994, was a Web directory called Yahoo! Directory.
  - In 1995, a search function was added, allowing users to search Yahoo! Directory. It became one of the most popular ways for people to find web pages of interest, but its search function operated on its web directory, rather than its full-text copies of web pages.
- Soon after, a number of search engines appeared and vied for popularity namely Magellan, Excite, Infoseek, Inktomi, Northern Light, and AltaVista.
- In 1996, Robin Li developed the RankDex site-scoring algorithm for search engines results page ranking.
  - Received a US patent for the technology.
  - It was the first search engine that used hyperlinks to measure the quality of websites it was indexing.
  - Li later used his Rankdex technology for the Baidu search engine, which was founded by him in China and launched in 2000.
- Similar algorithm patent filed and paper published by Google two years later in 1998 for PageRank.
- Source: Wikipedia

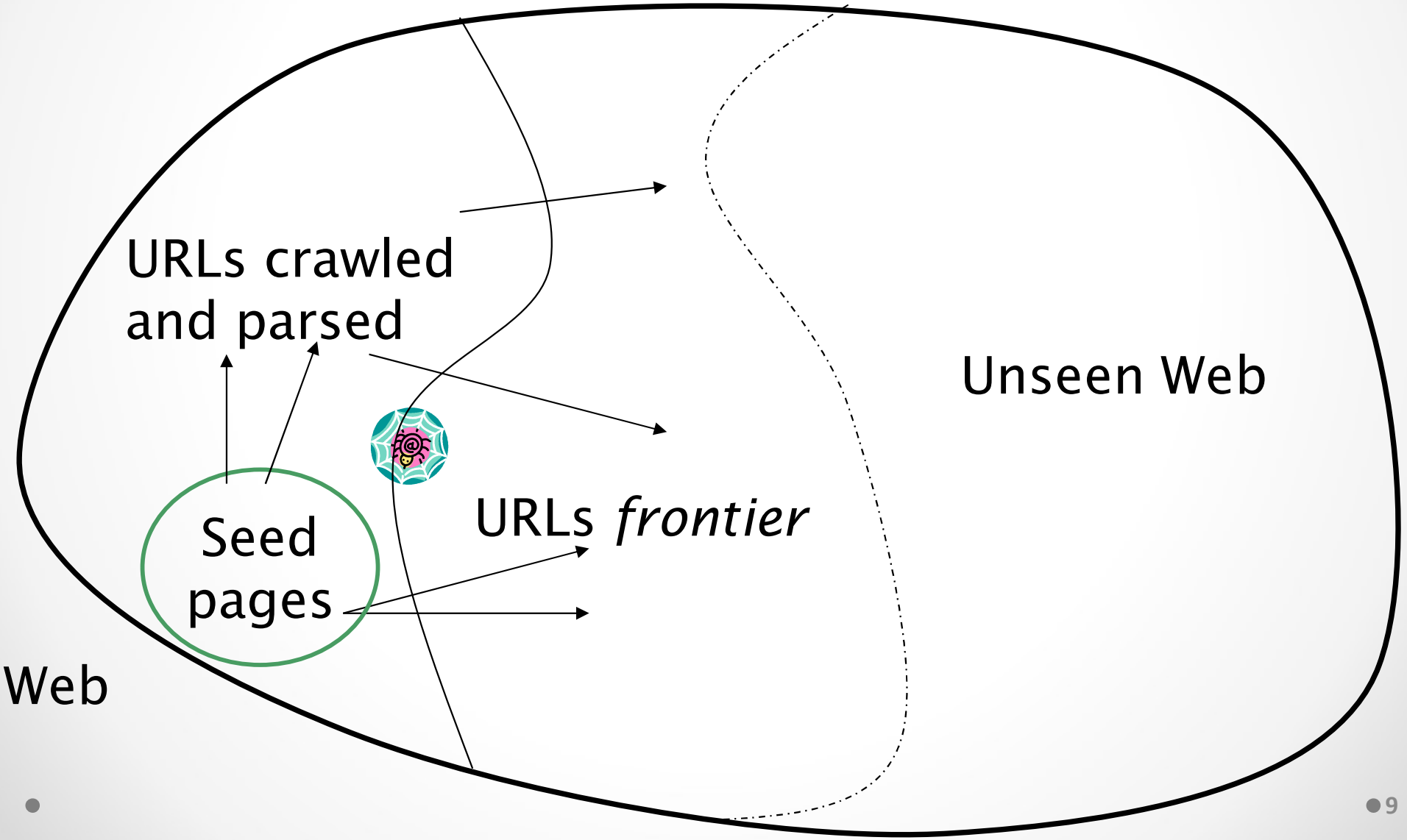
Overview

- Intro to Web Search
- Crawling
- Web Link Analysis
  - HITS
  - PageRank
- Beyond Page Rank
  - Trust Rank
  - HillTop Algorithm
- Link Analysis in other scenarios

Basic crawler operation

- Begin with known "seed" URLs
- Fetch and parse them
  - Extract URLs they point to
  - Place the extracted URLs on a queue
- Fetch each URL on the queue and repeat

Crawling picture



Complications

- Web crawling isn't feasible with one machine
  - All of the above steps distributed
- Malicious pages
  - Spam pages
  - Spider traps – incl dynamically generated
  - Recent Trends: Fake news, Hate propagation
- Even non-malicious pages pose challenges
  - Latency/bandwidth to remote servers vary
  - Webmasters' stipulations
    - How "deep" should you crawl a site's URL hierarchy?
  - Site mirrors and duplicate pages
- Politeness – don't hit a server too often

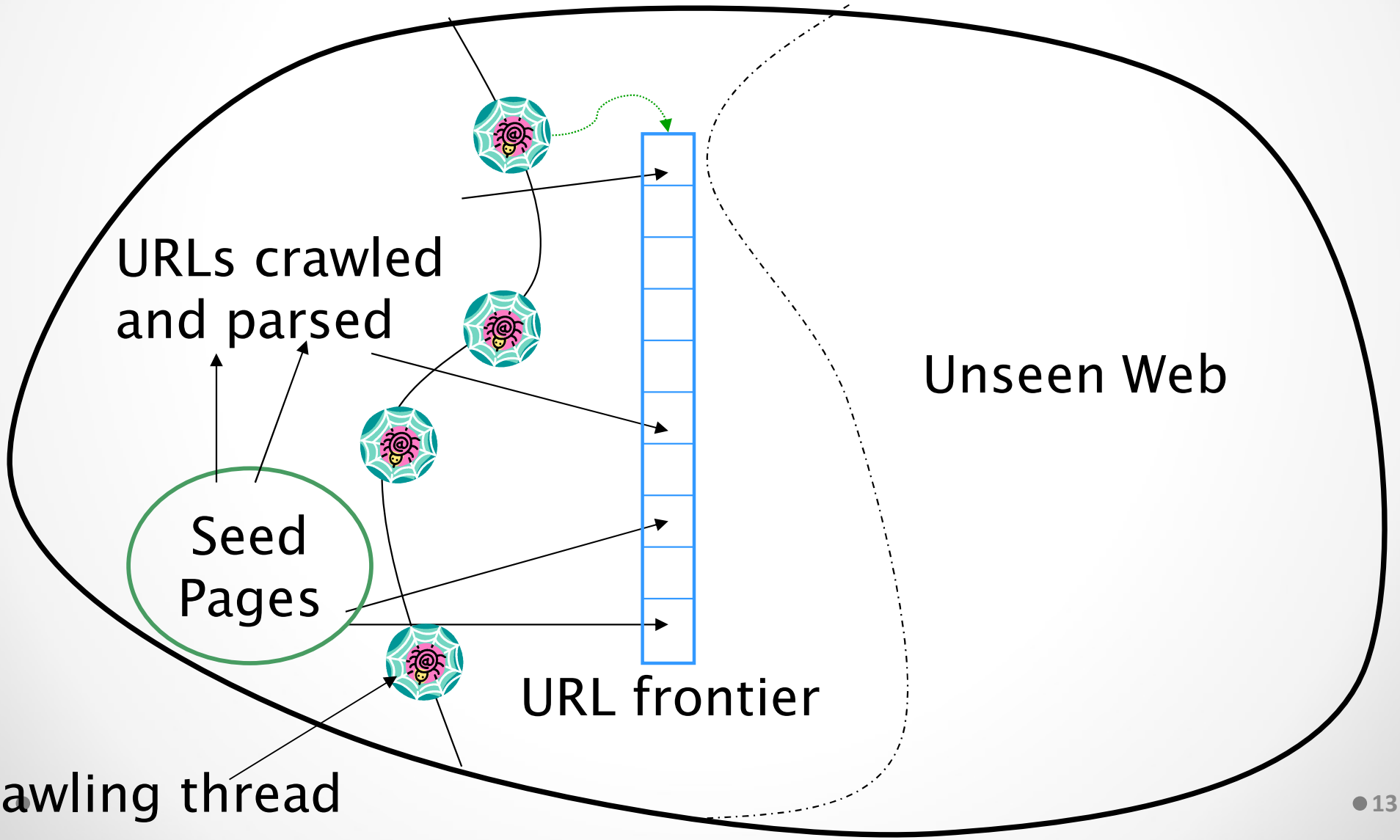
What any crawler *must* do

- Be Polite: Respect implicit and explicit politeness considerations
  - Only crawl allowed pages
  - Respect *robots.txt* (more on this shortly)
- Be Robust: Be immune to spider traps and other malicious behavior from web servers

What any crawler *should* do

- Be capable of distributed operation: designed to run on multiple distributed machines
- Be scalable: designed to increase the crawl rate by adding more machines
- Performance/efficiency: permit full use of available processing and network resources
- Fetch pages of "higher quality" first
- Continuous operation: Continue fetching fresh copies of a previously fetched page
- Extensible: Adapt to new data formats, protocols

Updated crawling picture



URL frontier

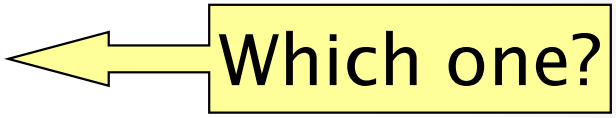
- Can include multiple pages from the same host
- Must avoid trying to fetch them all at the same time
- Must try to keep all crawling threads busy

Explicit and implicit politeness

- Explicit politeness: specifications from webmasters on what portions of site can be crawled
  - robots.txt
- Implicit politeness: even with no specification, avoid hitting any site too often

Processing steps in crawling

- Pick a URL from the frontier
- Fetch the document at the URL
- Parse the URL
  - Extract links from it to other docs (URLs)
- Check if URL has content already seen
  - If not, add to indexes
- For each extracted URL
  - Ensure it passes certain URL filter tests
  - Check if it is already in the frontier (duplicate URL elimination)



Breadth-First Traversal

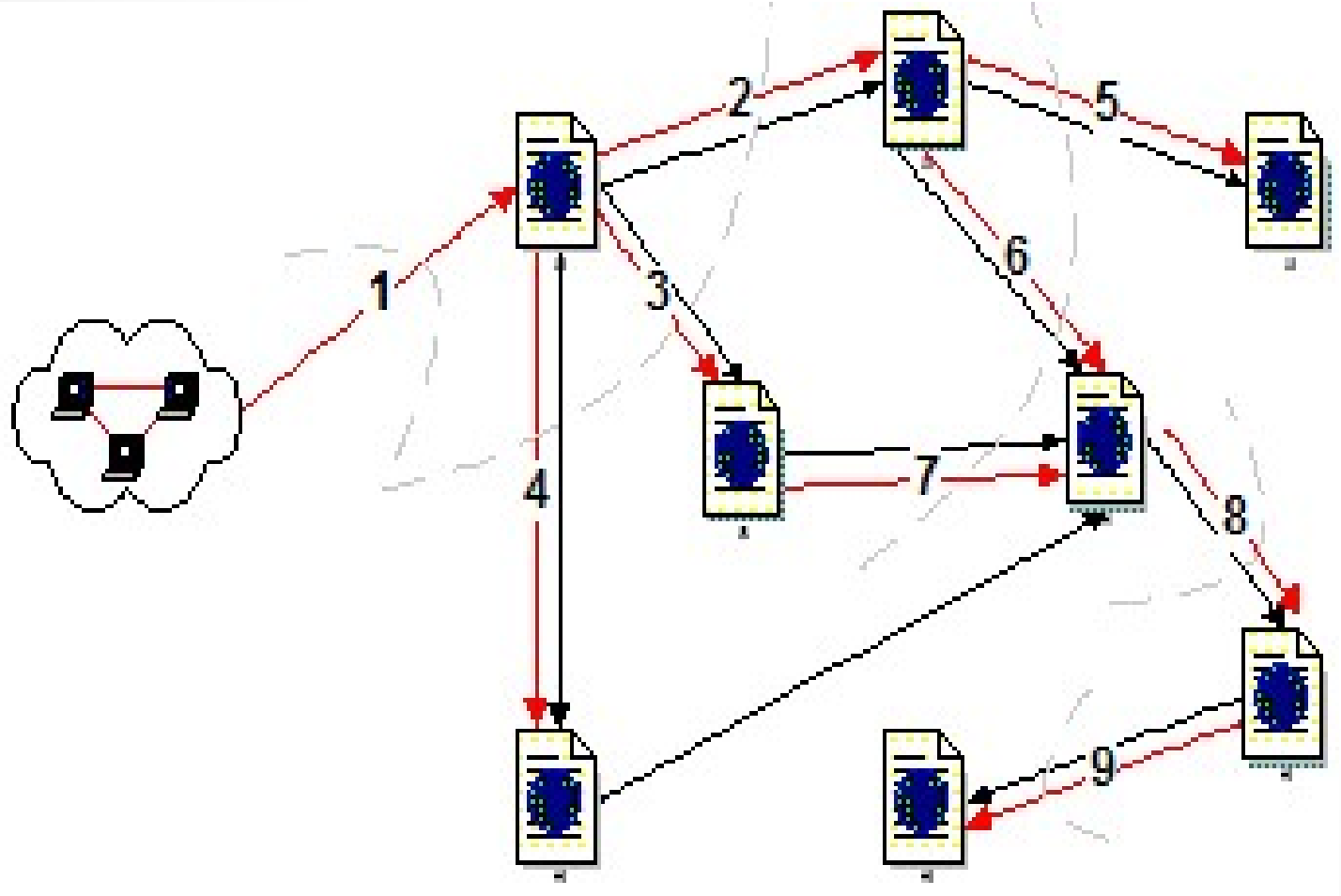
- Given any graph and a set of seeds at which to start, the graph can be traversed using the algorithm:

1. Put all the given seeds into the queue;
2. Prepare to keep a list of "visited" nodes (initially empty);
3. As long as the queue is not empty:
   a. Remove the first node from the queue;
   b. Append that node to the list of "visited" nodes
   c. For each edge starting at that node:

i. If the node at the end of the edge already appears on the list of "visited" nodes or it is already in the queue, then do nothing more with that edge;

ii. Otherwise, append the node at the end of the edge to the end of the queue.

Breadth First Crawlers

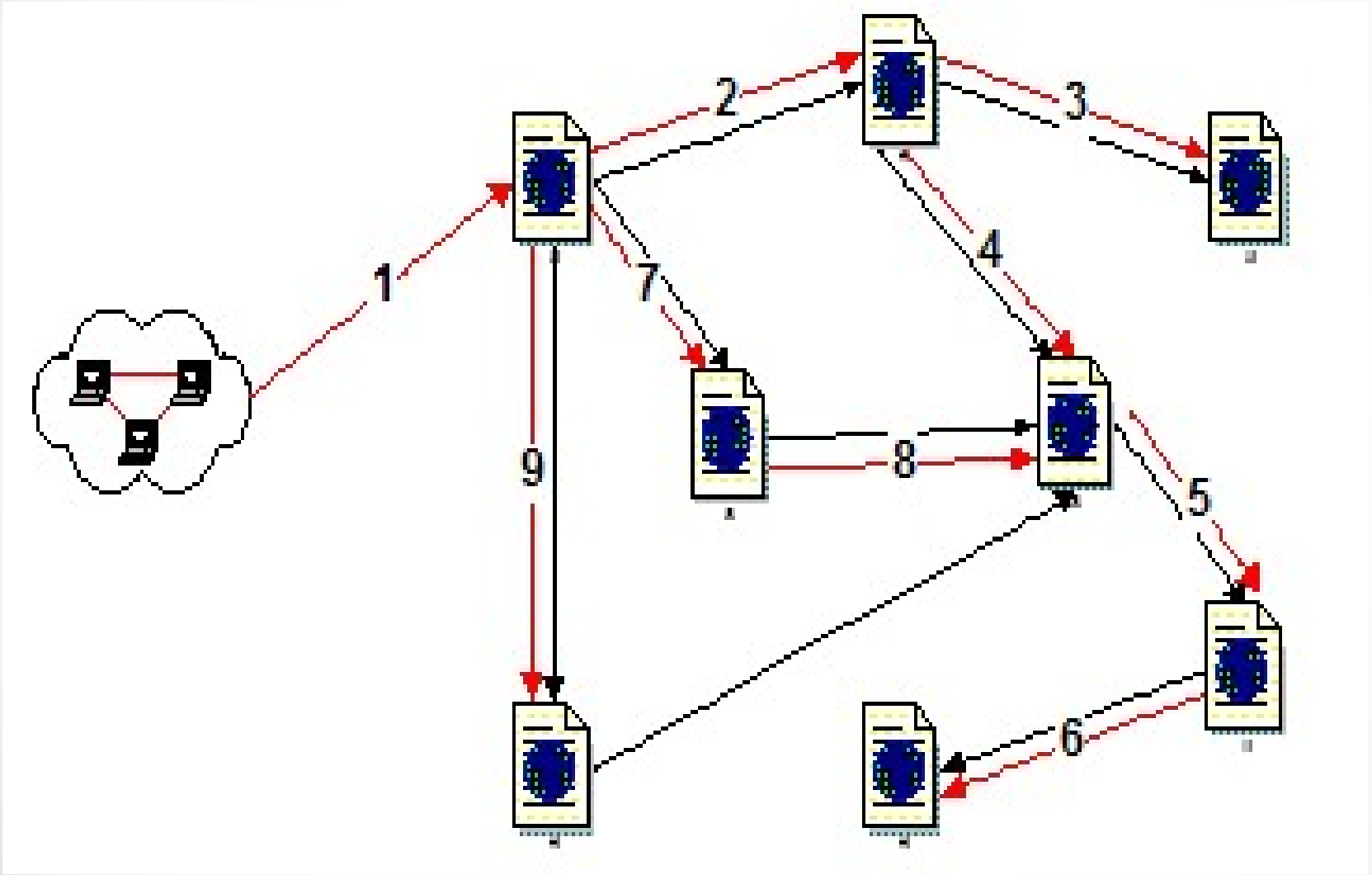


Depth First Crawlers

- Use depth first search (DFS) algorithm

1. Get the 1st link not visited from the start page
2. Visit link and get 1st non-visited link
3. Repeat above step till no non-visited links
4. Go to next non-visited link in the previous level and repeat 2nd step

Depth first traversal



Depth-First vs. Breadth-First

- Depth-first goes off into one branch until it reaches a leaf node
  - not good if the goal node is on another branch
  - neither complete nor optimal
  - uses much less space than breadth-first
    - much fewer visited nodes to keep track of
    - smaller fringe
- Breadth-first is more careful by checking all alternatives
  - complete and optimal
  - very memory-intensive

Overview

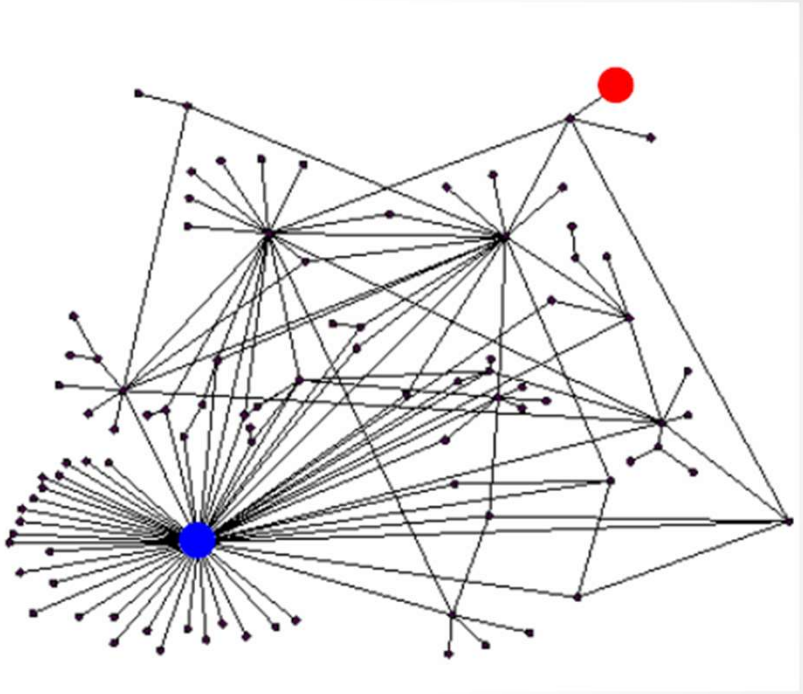
- Intro to Web Search
- Crawling
- Web Link Analysis
  - HITS
  - PageRank
  - Variations of Page Rank
- Beyond Page Rank
- Link Analysis in other scenarios

Overview

- Intro to Web Search
- Crawling
- Web Link Analysis
  - HITS
  - PageRank
  - Variations of Page Rank
- Beyond Page Rank
- Link Analysis in other scenarios

Ranking Nodes on the Graph

- **All web pages are not equally "important"**
  - thispersondoesnotexist.com vs. https://home.iitd.ac.in/
- There is large diversity in the web-graph node connectivity.
- **So, let's rank the pages using the web graph link structure!**



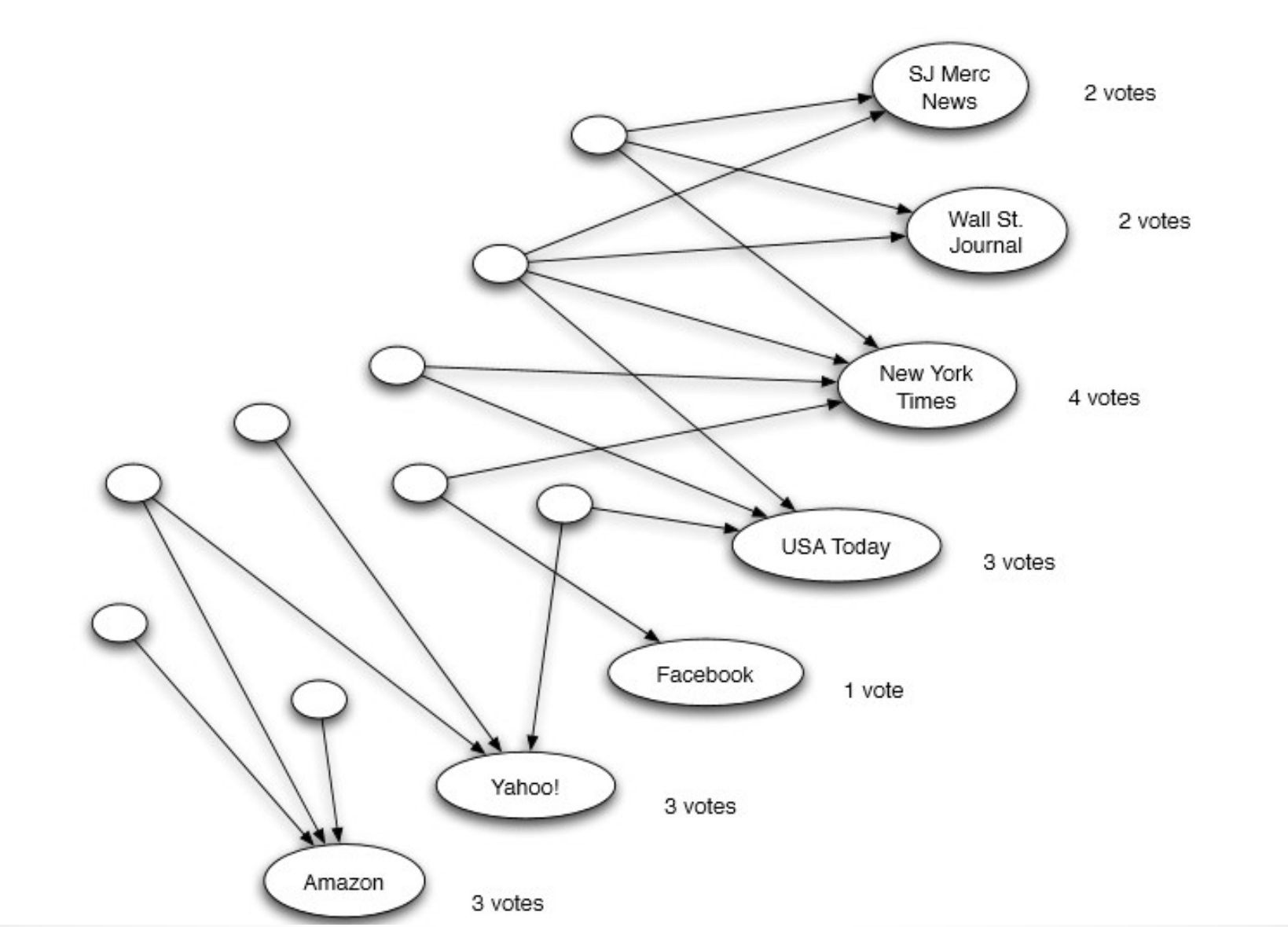
Hubs & Authorities

- Authorities:
  - Good source of information on a topic.
  - Good authority represented a page that was linked by many different hubs
- Hubs:
  - Large directories that were not actually authoritative in the information that they held
  - Were used as compilations of a broad catalog of information that led users direct to other authoritative pages.
  - In other words, a good hub represented a page that pointed to many other pages
- HITS Algorithm: To find good hubs & authorities [1]

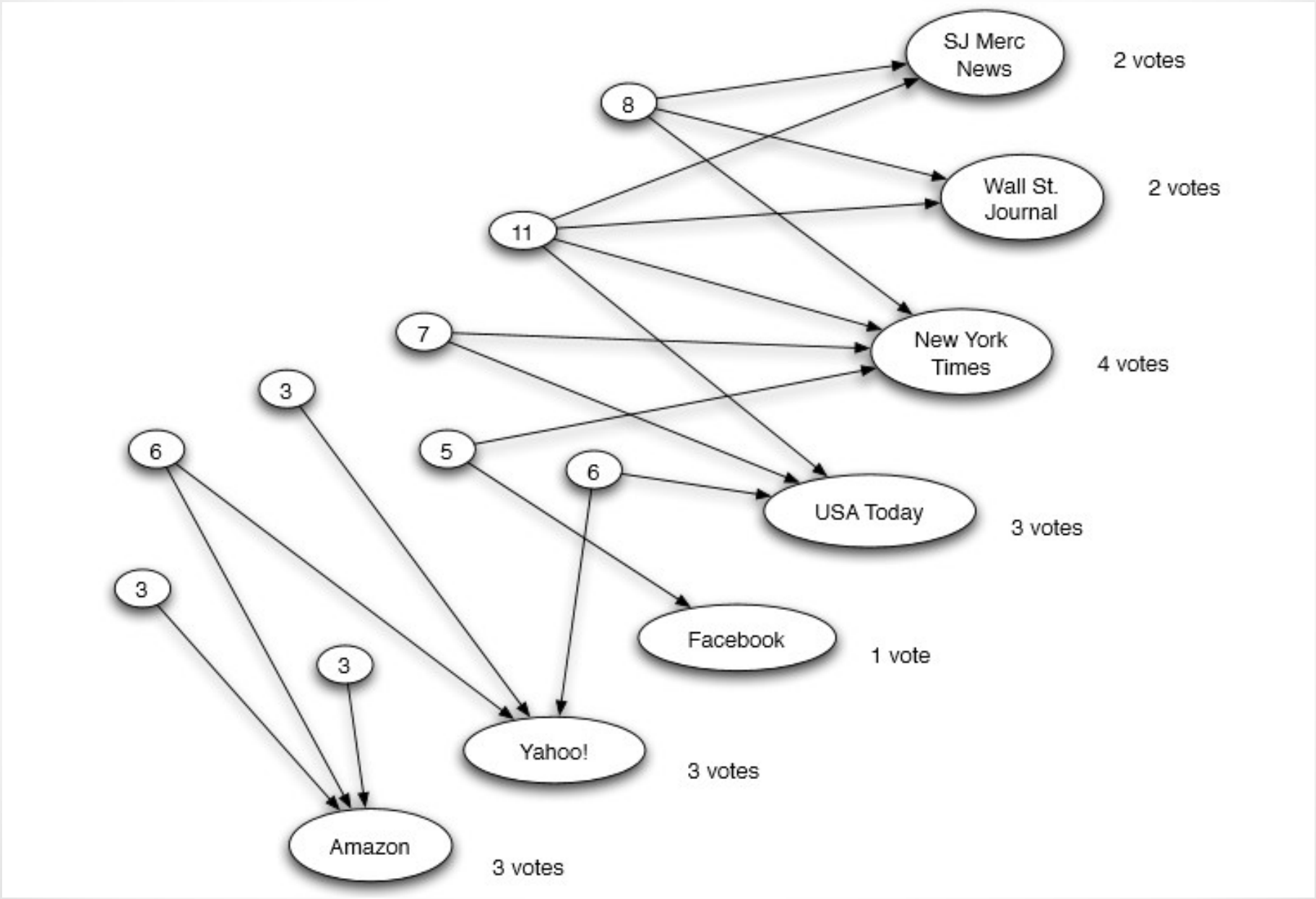
HITS Algorithm

- Retrieve the most relevant pages to the search query(*Root set*)
- Obtain a *base set* is generated by augmenting the root set with all the web pages that are linked from it and some of the pages that link to it.
- The web pages in the base set and all hyperlinks among those pages form a focused subgraph.
- Authority and hub values are defined in terms of one another in a mutual recursion.
  - An authority value is computed as the sum of the scaled hub values that point to that page.
  - A hub value is the sum of the scaled authority values of the pages it points to. Some implementations also consider the relevance of the linked pages.
- The algorithm performs a series of iterations, each consisting of two basic steps:
  - **Authority Update**: Update each node's *Authority score* to be equal to the sum of the *Hub Scores* of each node that points to it. That is, a node is given a high authority score by being linked from pages that are recognized as Hubs for information.
  - **Hub Update**: Update each node's *Hub Score* to be equal to the sum of the *Authority Scores* of each node that it points to. That is, a node is given a high hub score by linking to nodes that are considered to be authorities on the subject.

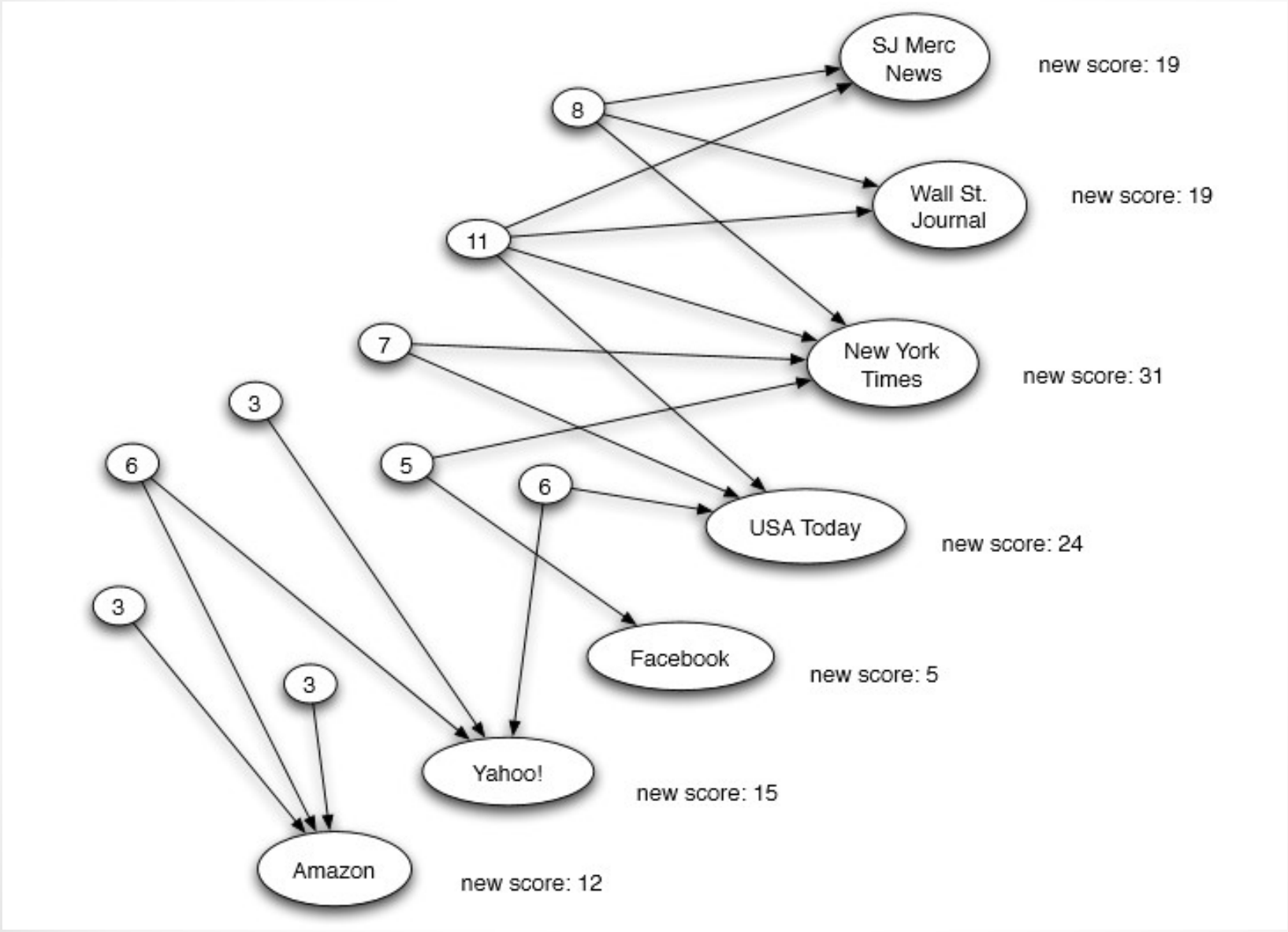
HITS Algorithm – 1st Authority Update



HITS Algorithm – 1st Hub Update



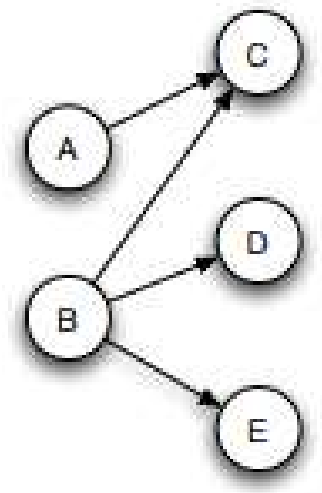
HITS Algorithm – 2nd Authority Update



HITS Algorithm - Steps

- The Hub score and Authority score for a node is calculated with the following algorithm:
  I. Start with each node having a hub score and authority score of 1.
  II. Run the Authority Update Rule
  III. Run the Hub Update Rule
  IV. Normalize the values by dividing each Hub score by square root of the sum of the squares of all Hub scores and dividing each Authority score by square root of the sum of the squares of all Authority scores.
  V. Repeat from the second step as necessary *k* times or until the value changes are very small

HITS Algorithm – Exercise

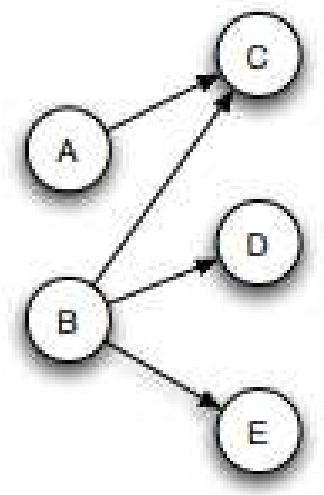


Hub & Authority Scores after 2 iterations of HITS Algorithm

HITS Algorithm – Exercise

HUBS

A	B
2	4
6	14



AUTHORITIES

C	D	E
2	1	1
6	4	4

Hub & Authority Scores after 2 iterations of HITS Algorithm

HITS Algorithm - Features

- Works on *Principle of Repeated Improvement*
- The normalized values actually converge to limits as k goes to infinity
- Except in a few rare cases (characterized by a certain kind of degenerate property of the link structure), we reach the same limiting values no matter what we choose as the initial hub and authority values, provided only that all of them are positive
  - The limiting hub and authority values are a property purely of the link structure, not of the initial estimates

Overview

- Intro to Web Search
- Crawling
- Web Link Analysis
  - HITS
  - PageRank
  - Variations of Page Rank
- Beyond Page Rank
- Link Analysis in other scenarios

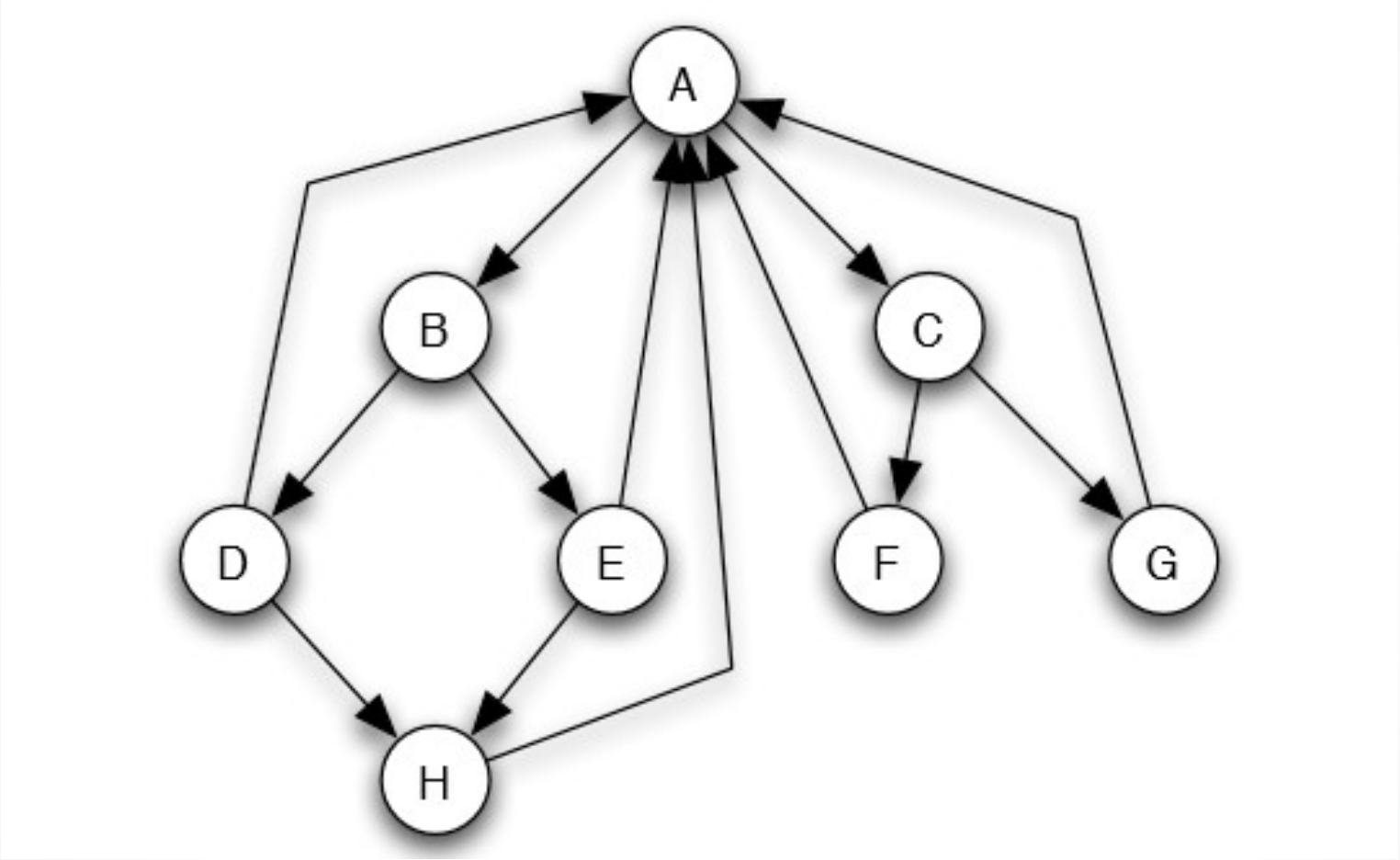
Link as Votes

- **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:**
- **Are all in-links equal?**
  - Links from important pages count more
  - Recursive question!
- **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 [2]

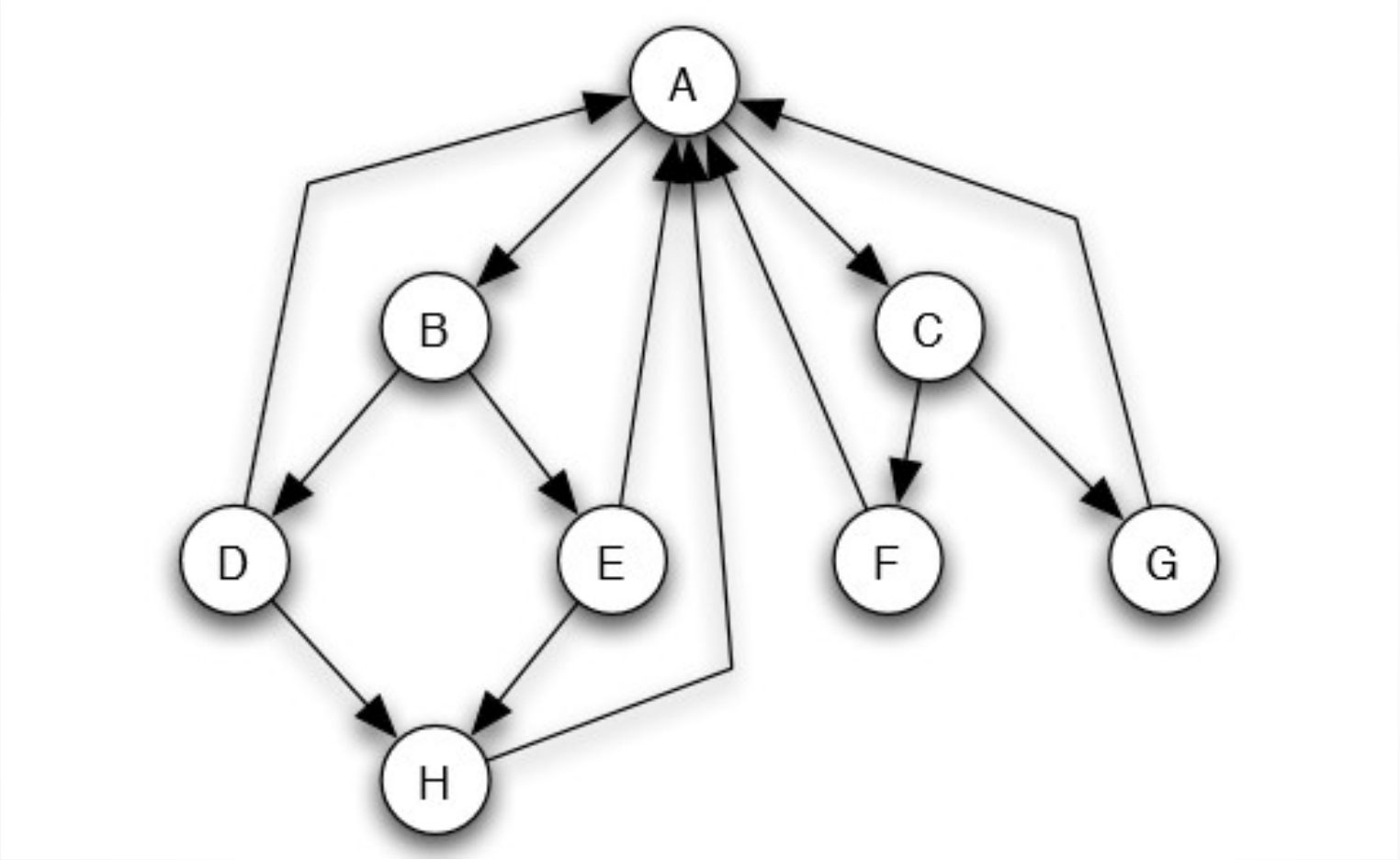
Page Rank

- In a network with *n* nodes, we assign all nodes the same initial PageRank, set to be *1/n*
- We then perform a sequence of k updates to the PageRank values, using the following rule for each update:
  - Each page divides its current PageRank equally across its out-going links and passes these equal shares to the pages it points to. (If a page has no out-going links, it passes all its current PageRank to itself.)
  - Each page updates its new PageRank to be the sum of the shares it receives

Exercise – PageRank – Initial Graph

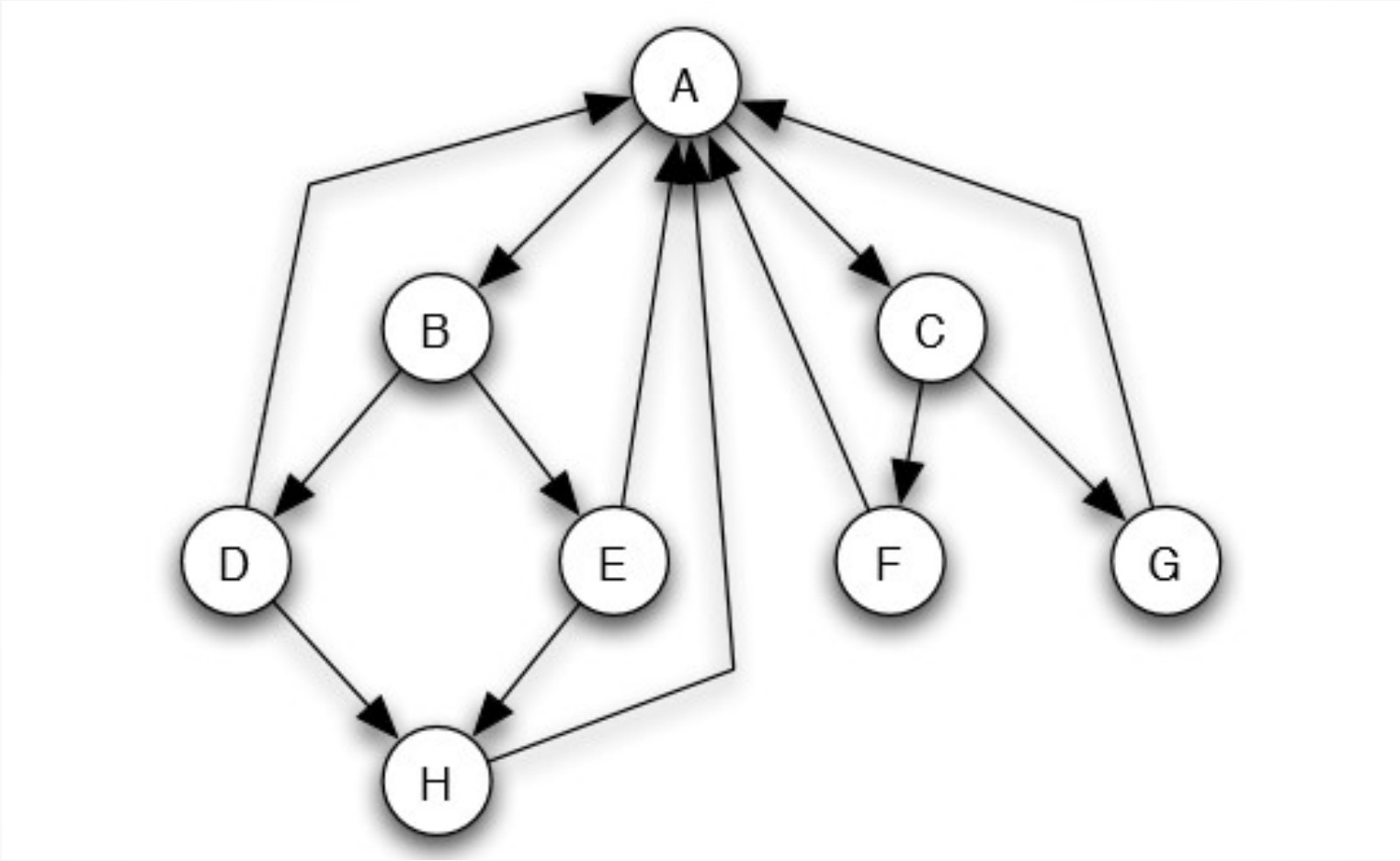


Exercise – PageRank after 1 update



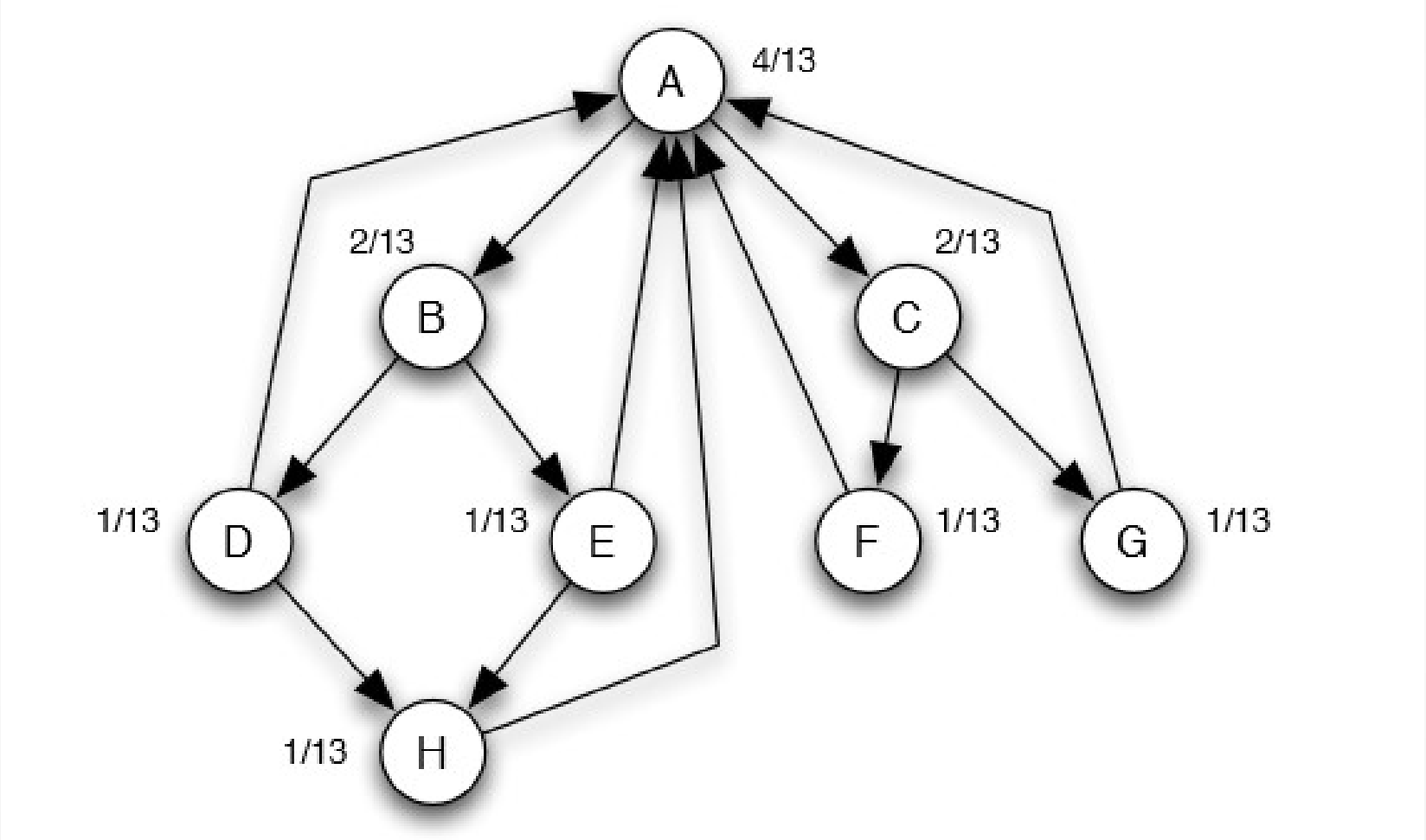
Step	A	B	C	D	E	F	G	H
1	1/2	1/16	1/16	1/16	1/16	1/16	1/16	1/8

Exercise – PageRank after 2 updates



Step	A	B	C	D	E	F	G	H
1	1/2	1/16	1/16	1/16	1/16	1/16	1/16	1/8
2	5/16	1/4	1/4	1/32	1/32	1/32	1/32	1/16

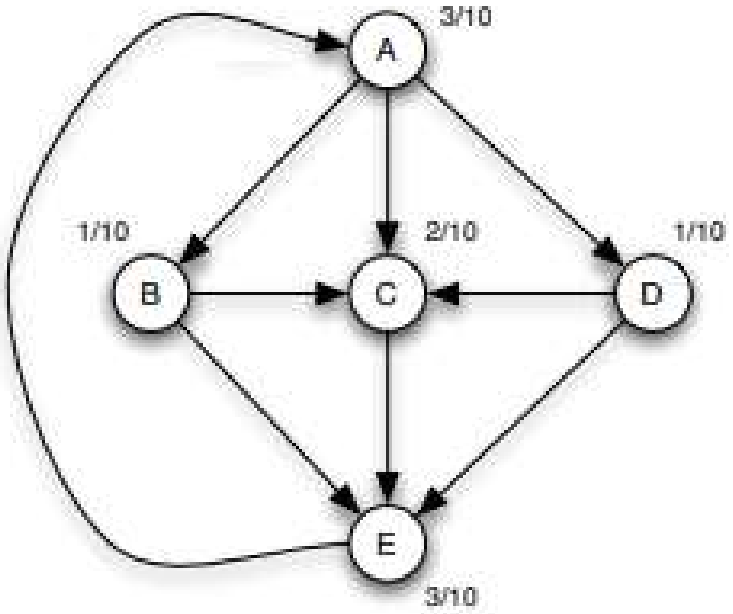
Final PageRank



Page Rank - Features

- The values actually converge to limits as *k* goes to infinity
- PageRank is conserved throughout the computation — with the total PageRank in the network equal to one
- Applying one step of the Basic PageRank Update Rule the limiting PageRank values and, then the values at every node remain the same
- If the network is strongly connected then there is a unique set of equilibrium value

Exercise



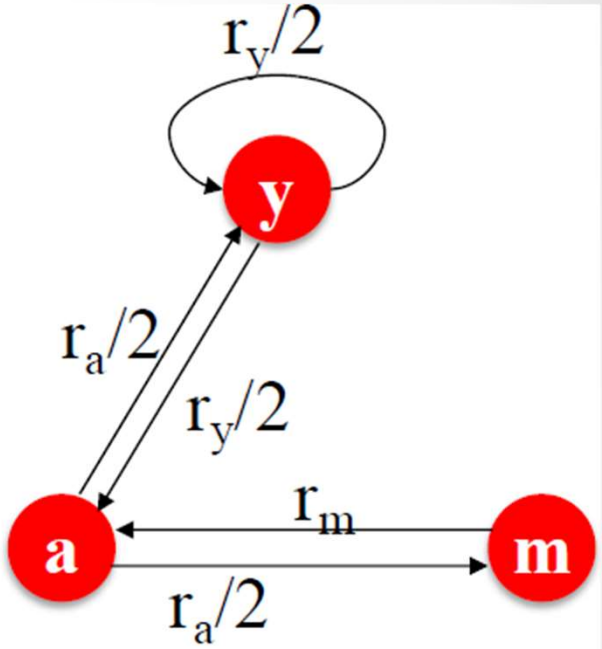
Are these the correct final PageRank values for this network?

Page Rank: Flow Model

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

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

d_i … out-degree of node i



"Flow" equations:

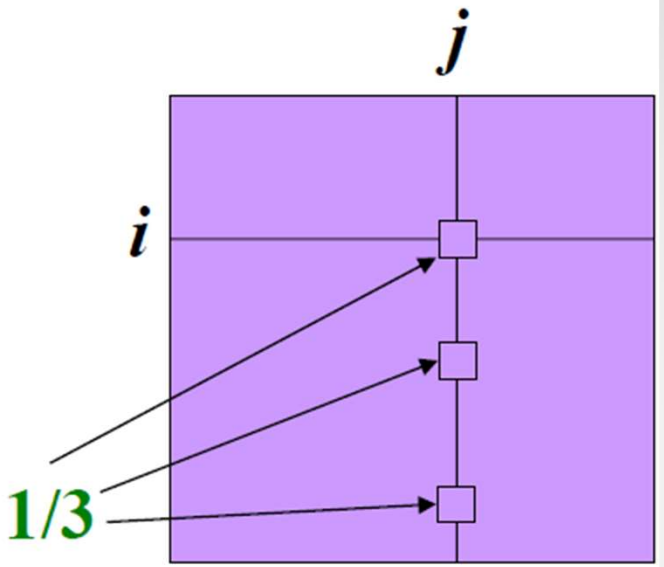
$r_y = r_y/2 + r_a/2$

$r_a = r_y/2 + r_m$

$r_m = r_a/2$

Page Rank: 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**



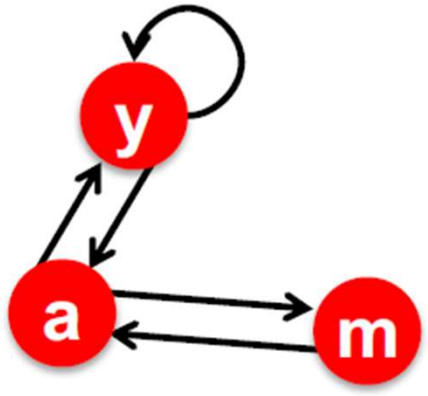
- **Rank vector r:** An entry per page
  - r_i is the importance score of page i
  - $\sum_i r_i = 1$
- **The flow equations can be written**

$$r = M \cdot r$$

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

M

Flow Equations & M



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

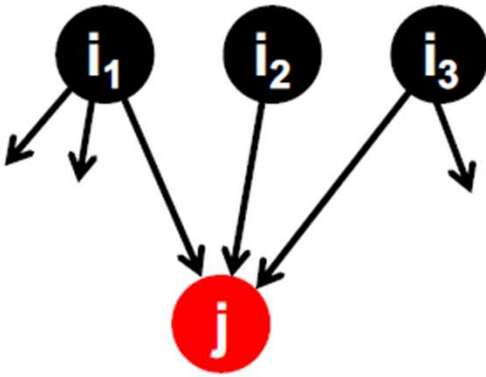
$$r_y = r_y/2 + r_a/2$$
$$r_a = r_y/2 + r_m$$
$$r_m = r_a/2$$

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

r M r

Connection to Random Walk

- **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



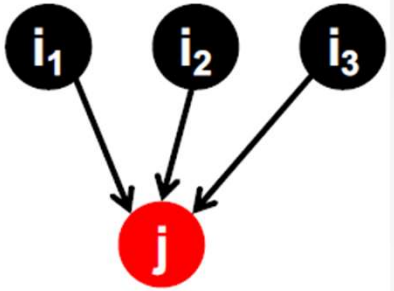
$$r_j = \sum_{i \to j} \frac{r_i}{\mathrm{d}_{\mathrm{out}}(\mathrm{i})}$$

- **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

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

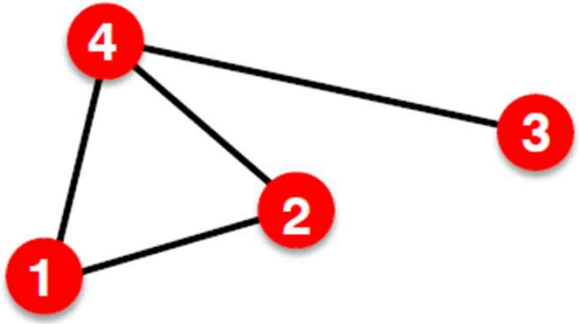
$$\boldsymbol{p(t+1) = M \cdot p(t) = p(t)}$$

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

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

Eigenvector of a Matrix

$\boldsymbol{A \in \{0,1\}^{n \times n}}$ be an adj. matrix of undir. graph:



$$A = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

- Eigenvector of adjacency matrix: vectors satisfying $\boldsymbol{\lambda c = Ac}$
- $\boldsymbol{c}$: eigenvector; λ: eigenvalue

- Note:
  - This is the definition of eigenvector centrality (for undirected graphs).
  - PageRank is defined for directed graphs

Eigenvector Formulation

- **The flow equation:**

$$1 \cdot \boldsymbol{r} = \boldsymbol{M} \cdot \boldsymbol{r}$$

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

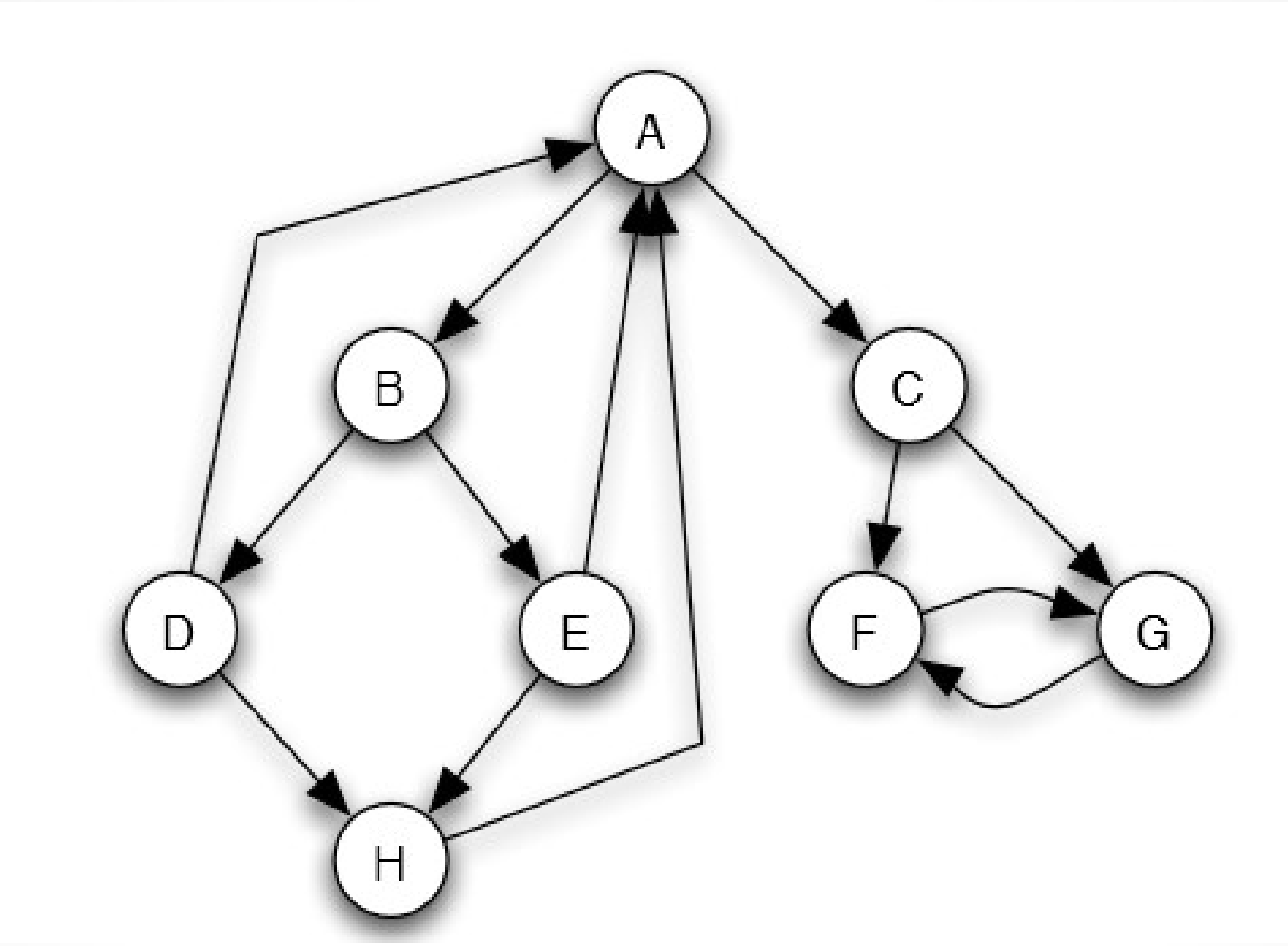
$\boldsymbol{r}$ $\quad$ $\boldsymbol{M}$ $\quad$ $\boldsymbol{r}$

- So the **rank vector** $\boldsymbol{r}$ is an **eigenvector** of the stochastic ajd. matrix $\boldsymbol{M}$ (with eigenvalue 1)
  - 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.
    - **PageRank** = Limiting distribution = **principal eigenvector** of M
    - Note: If $\boldsymbol{r}$ is the limit of the product $\boldsymbol{MM} \ldots \boldsymbol{Mu}$, then $\boldsymbol{r}$ satisfies the **flow equation** $1 \cdot \boldsymbol{r} = \boldsymbol{Mr}$
    - So $\boldsymbol{r}$ is the **principal eigenvector** of $\boldsymbol{M}$ with eigenvalue 1

Page Rank: Many Interpretations

- Measures importance of nodes in a graph using the link structure of the web
- Models a random web surfer using the stochastic adjacency matrix$\boldsymbol{M}$
- PageRank solves $\boldsymbol{r} = \boldsymbol{M}\boldsymbol{r}$ where $\boldsymbol{r}$ can be viewed as both the principle eigenvector of $\boldsymbol{M}$ and as the stationary distribution of a random walk over the graph
- The probability of being at a page X after k steps of this random walk is precisely the PageRank of X after k applications of the Basic PageRank Update Rule

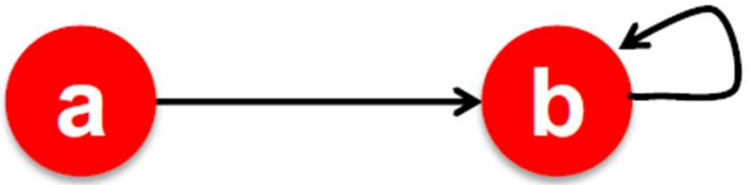
Problems with basic Page Rank



1. **Spider traps**
2. **Dead ends**

Spider Trap Problem

- All out-links are within the group
- Eventually spider traps absorb all importance



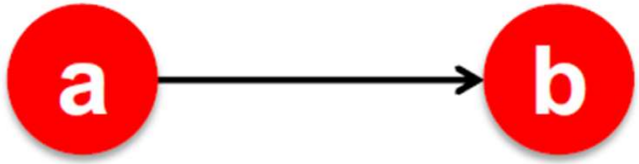
$$r_j^{(t+1)} = \sum_{i \to j} \frac{r_i^{(t)}}{d_i}$$

- **Example:**

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

Dead-end Problem

- Have no out-links)
- Such pages cause importance to "leak out"



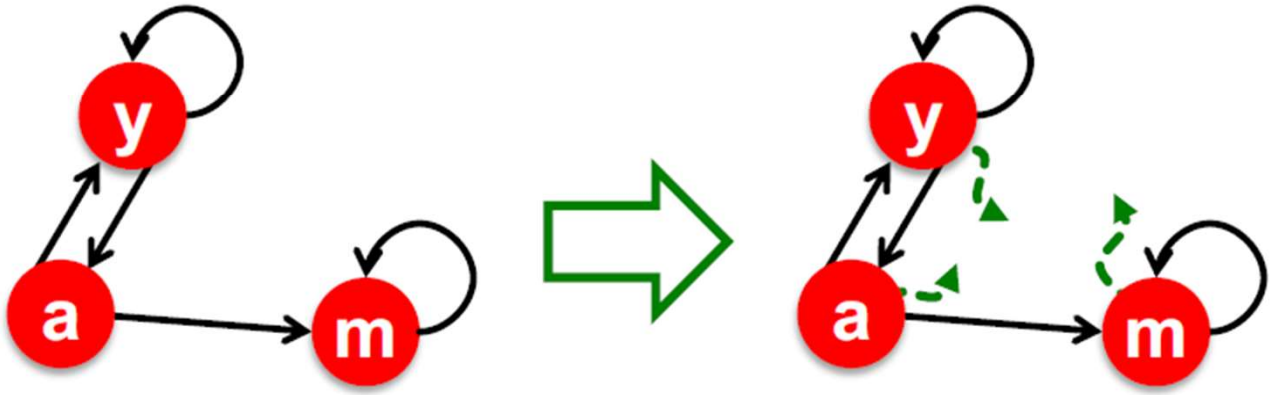
$$r_j^{(t+1)} = \sum_{i \to j} \frac{r_i^{(t)}}{d_i}$$

- **Example:**

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

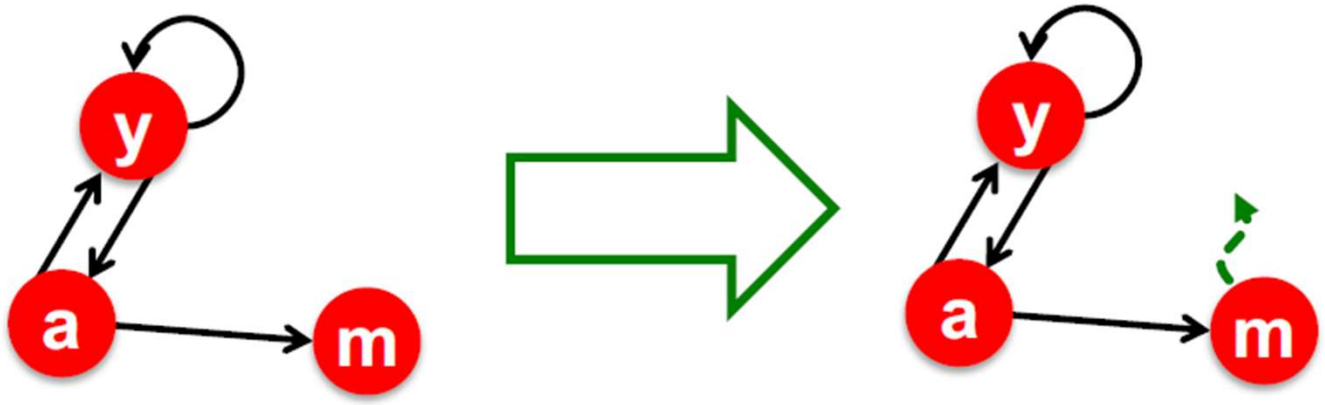
Solution to Spider Trap Problem

- **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-end Problem

- **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

Scaled Page Rank

- Solution to both the problems
- Scaled PageRank Update Rule:
  - First apply the Basic PageRank Update Rule.
  - Then scale down all PageRank values by a factor of *s*. This means that the total PageRank in the network has shrunk from *1* to *s*
  - We divide the residual *1−s* units of PageRank equally over all nodes, giving *(1−s)/n* to each
- In practice *s=0.8-0.9*

Scaled Page Rank & Random Walk

- With probability *s* the walker follows a random edge as before

- With probability *1–s* the walker jumps to a random node anywhere in the network, choosing each node with equal probability

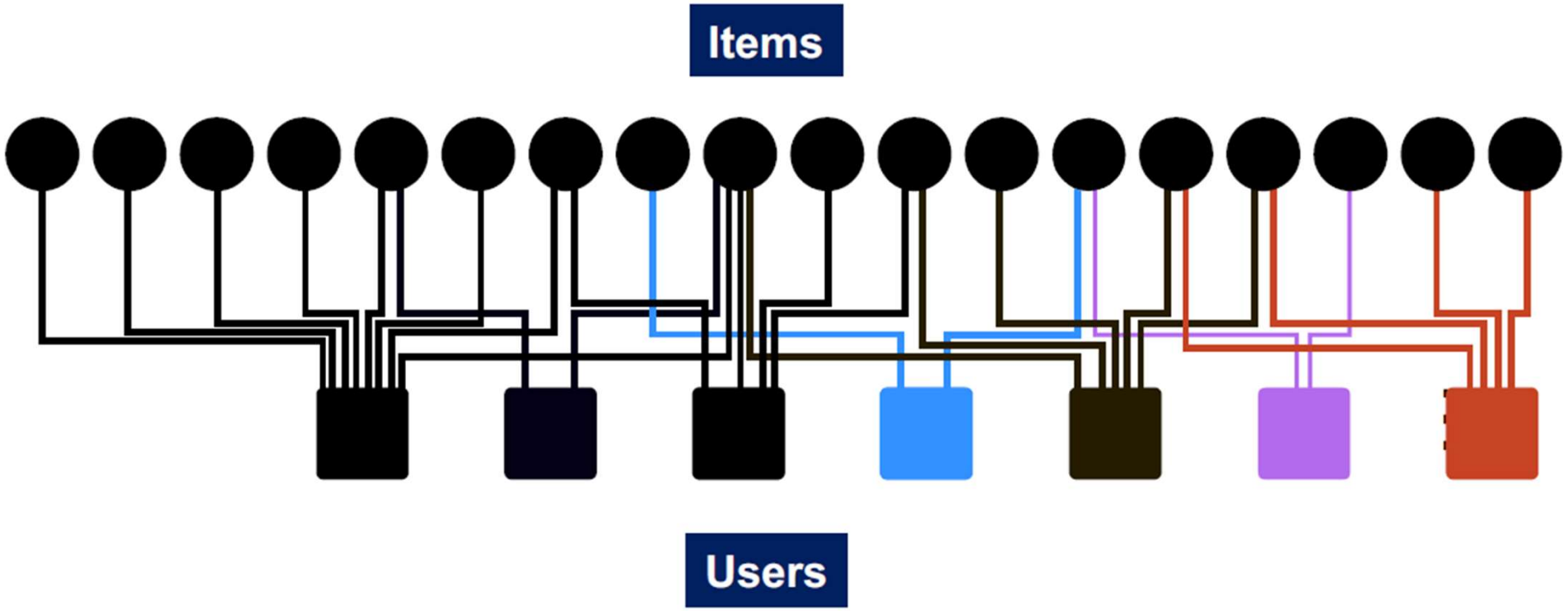
Overview

- Intro to Web Search
- Crawling
- Web Link Analysis
  - HITS
  - PageRank
  - Variations of Page Rank
- Beyond Page Rank
- Link Analysis in other scenarios

Recommendations

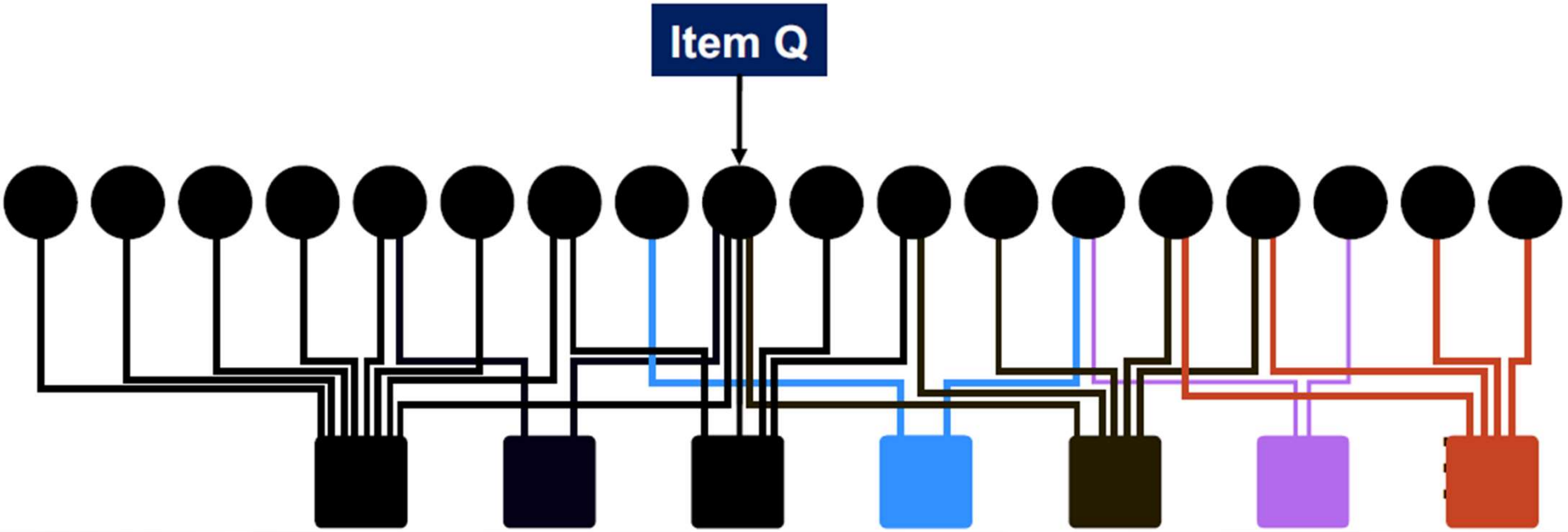
- **Given:**

  A bipartite graph representing user and item interactions (e.g. purchase)



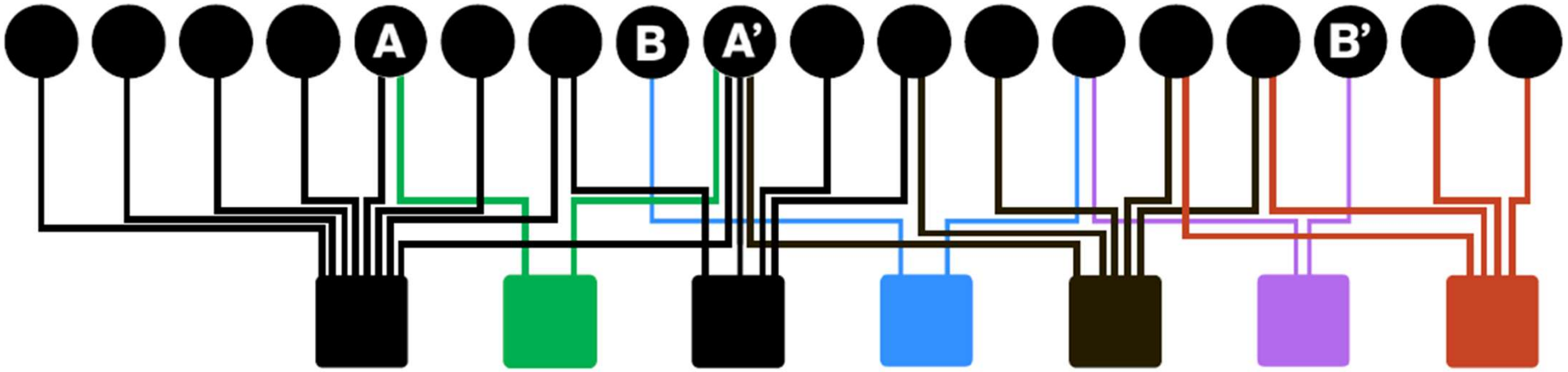
Bi-partite Item User Graph

- **Goal:** Proximity on graphs
  - **What items should we recommend to a user who interacts with item Q?**
  - **Intuition**: if items Q and P are interacted by similar users, recommend P when user interacts with Q



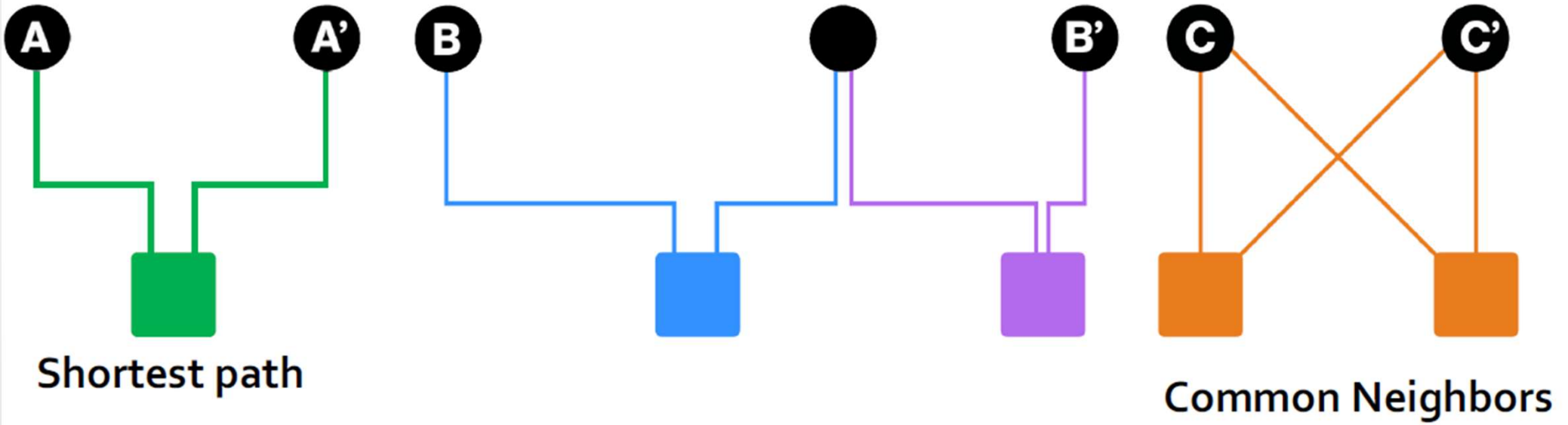
Bi-partite Item User Graph

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



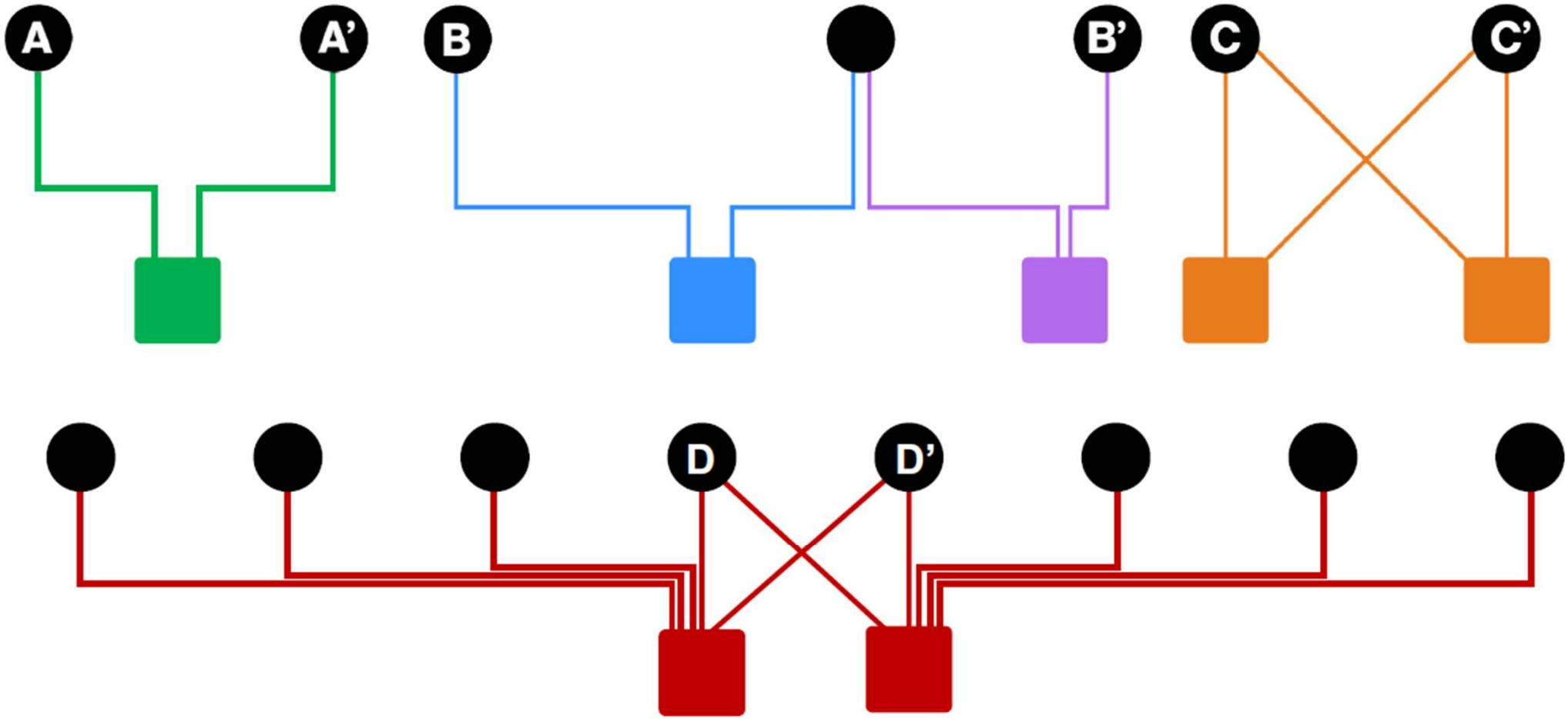
Node Proximity Measurements

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



Node Proximity Measurements

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



Proximity on Graphs

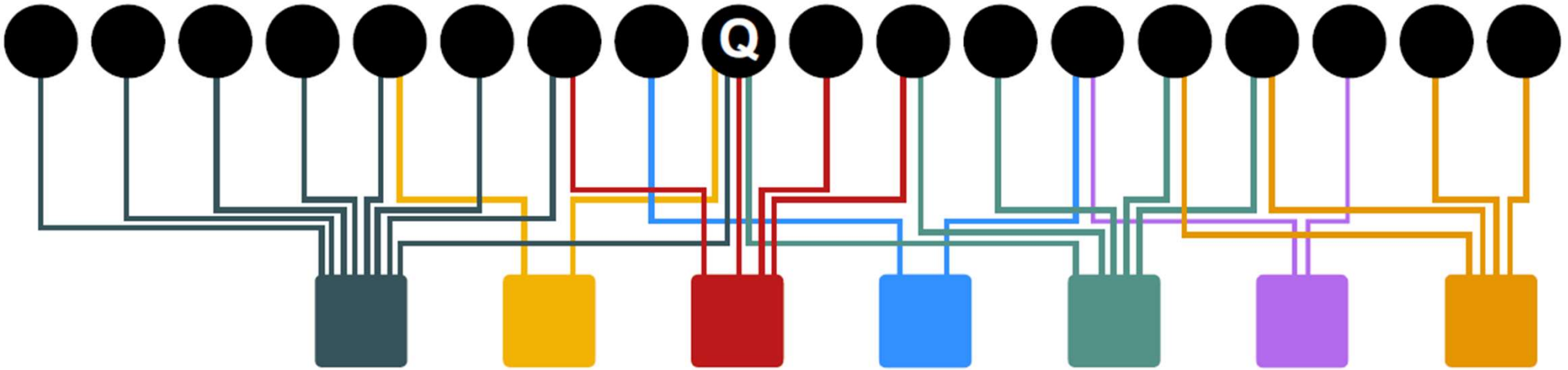
- **PageRank:**
  - Ranks nodes by "importance"
  - Teleports with uniform probability to any node in the network
- **Personalized PageRank:**
  - Ranks proximity of nodes to the teleport nodes $\boldsymbol{S}$
- **Proximity on graphs:**
  - **Q:** What is most related item to **Item Q**?
  - **Random Walks with Restarts**
    - Teleport back to the starting node: $\boldsymbol{S} = \{\boldsymbol{Q}\}$

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:

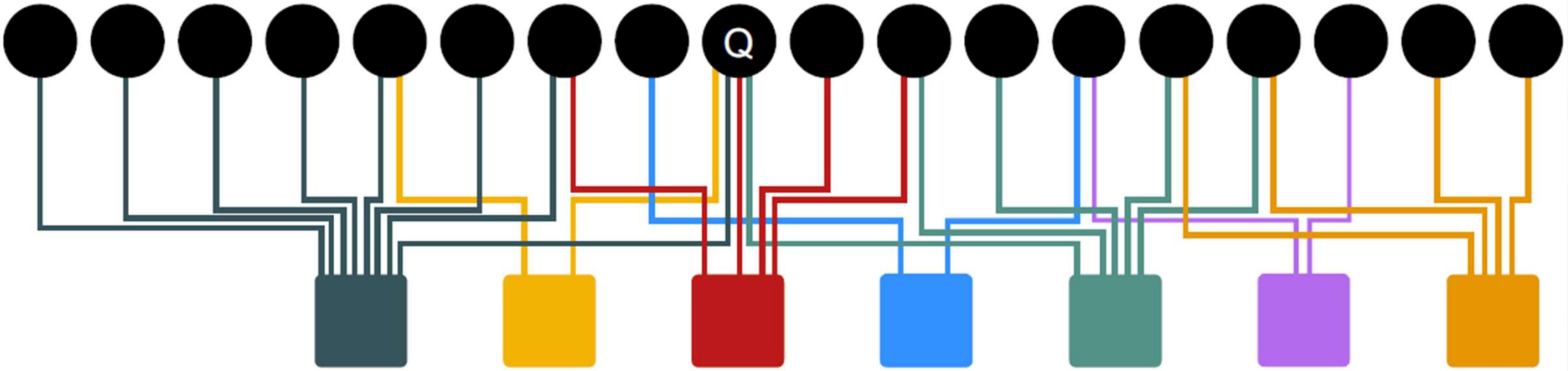


Random Walk Algorithm

- Proximity to query node(s) Q:

```
ALPHA = 0.5
QUERY_NODES = { Q }

item = QUERY_NODES.sample_by_weight( )
for i in range( N_STEPS ):
    user = item.get_random_neighbor( )
    item = user.get_random_neighbor( )
    item.visit_count += 1
    if random( ) < ALPHA:
        item = QUERY_NODES.sample.by_weight ( )
```



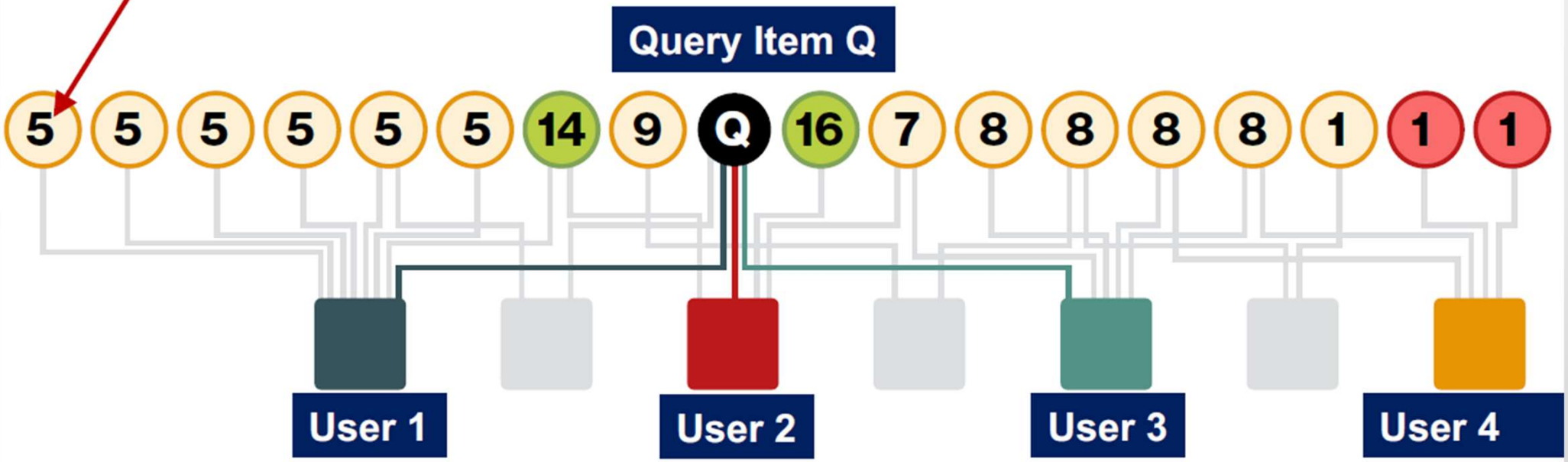
Random Walk Algorithm

- Proximity to query node(s) Q:

```
ALPHA = 0.5
QUERY_NODES = { Q }
```

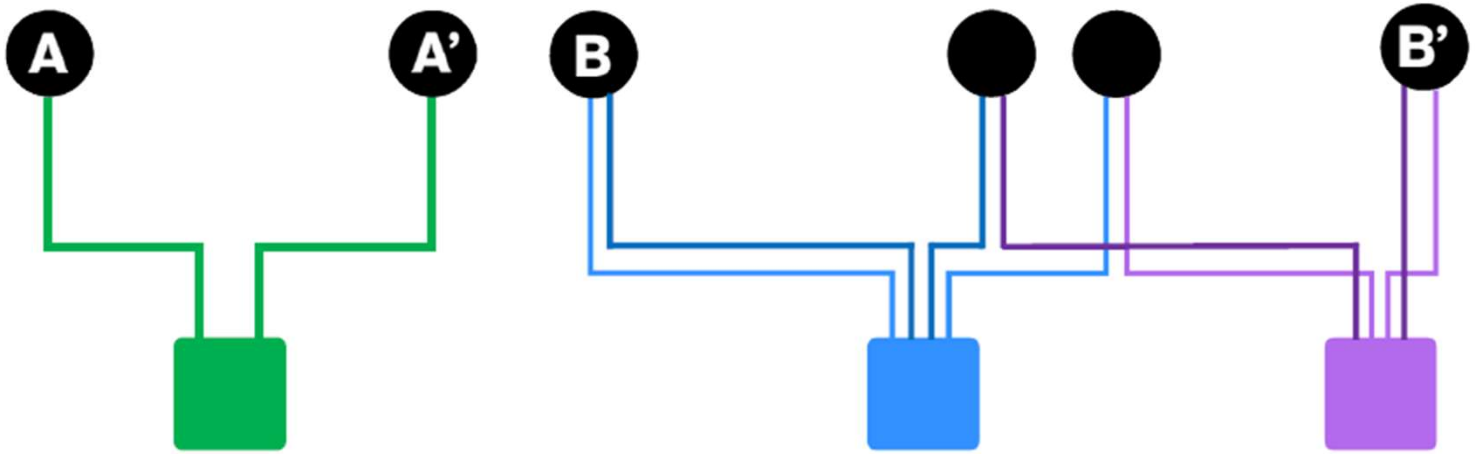
```
item = QUERY_NODES.sample_by_weight( )
for i in range( N_STEPS ):
    user = item.get_random_neighbor( )
    item = user.get_random_neighbor( )
    item.visit_count += 1
    if random( ) < ALPHA:
        item = QUERY_NODES.sample.by_weight ( )
```

Number of visits by random walks starting at Q



Benefits

- **Why is this a good solution?**
- **Because the "similarity" considers:**
  - Multiple connections
  - Multiple paths
  - Direct and indirect connections
  - Degree of the node



Page Rank Variants

- **PageRank:**
  - Teleports to any node
  - Nodes can have the same probability of the surfer landing:

$$\boldsymbol{S} = [0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1]$$

- **Topic-Specific PageRank** aka **Personalized PageRank:**
  - Teleports to a specific set of nodes
  - Nodes can have different probabilities of the surfer landing there:

$$\boldsymbol{S} = [0.1, 0, 0, 0.2, 0, 0, 0.5, 0, 0, 0.2]$$

- **Random Walk with Restarts:**
  - Topic-Specific PageRank where teleport is always to the same node:

$$\boldsymbol{S} = [0, 0, 0, 0, \mathbf{1}, 0, 0, 0, 0, 0, 0]$$