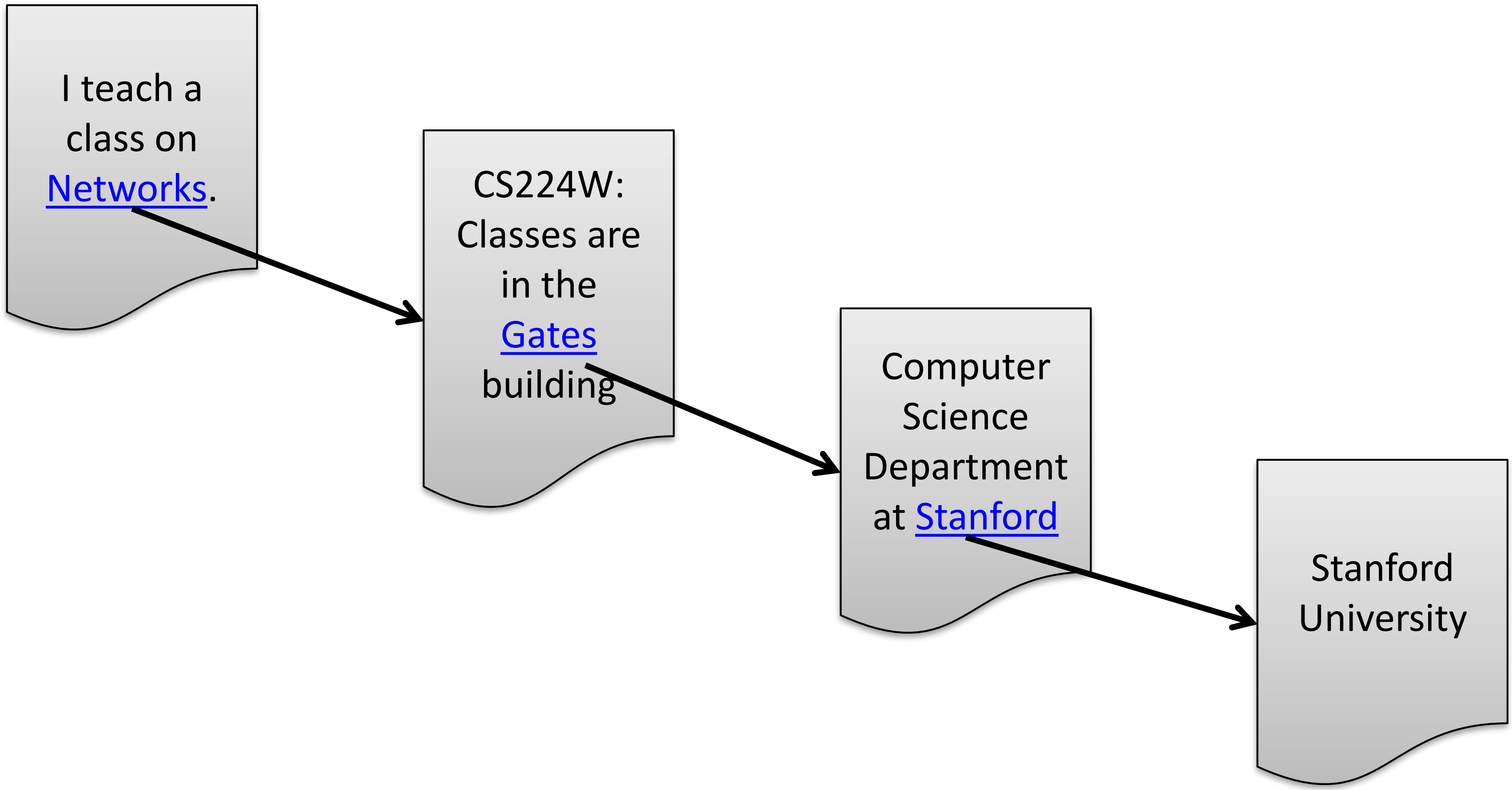
# Chapter 5

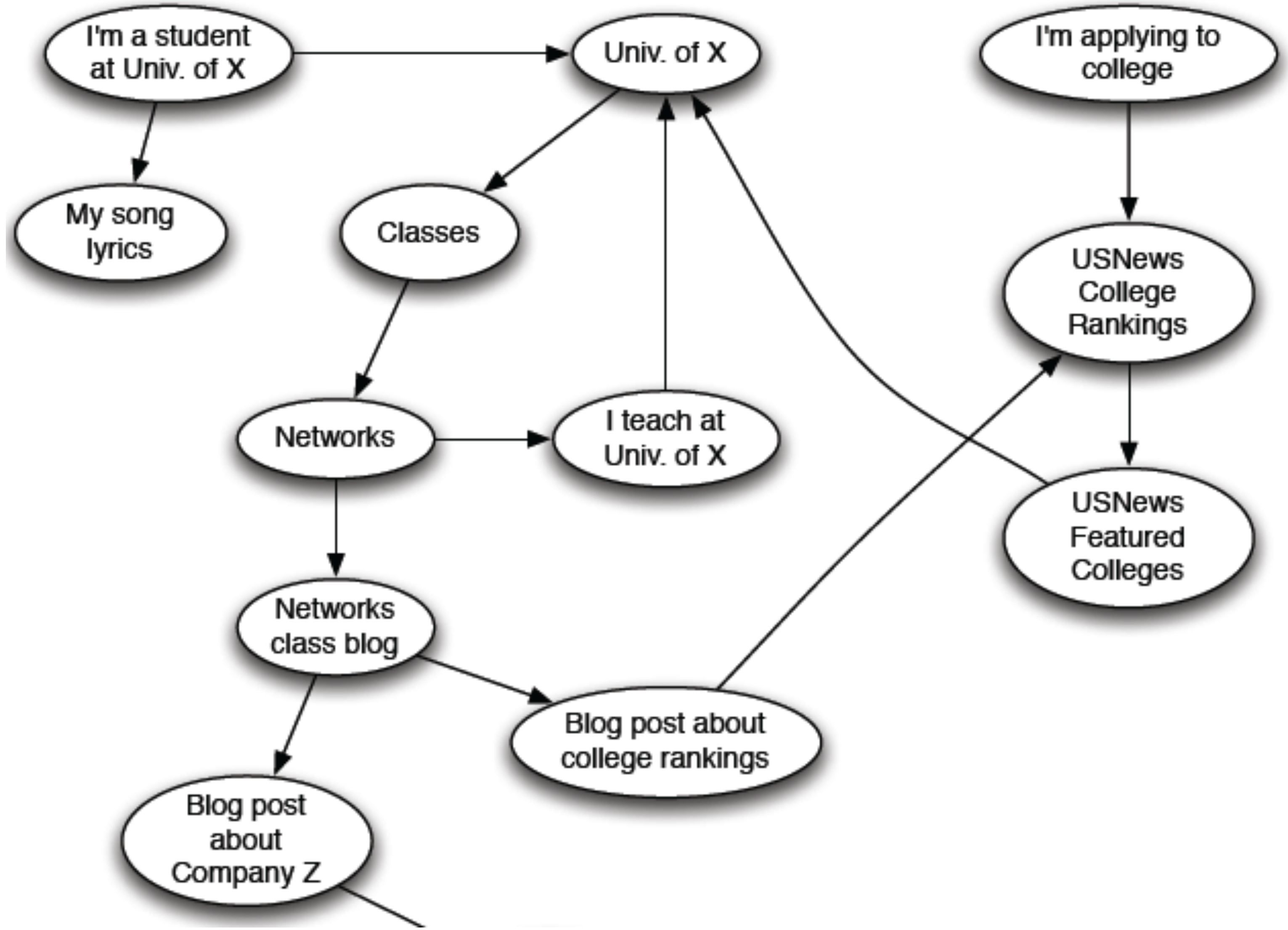
Link Analysis

# Web as a Graph

- We can see the Web as a ***directed graph***
  - Nodes: Web pages
  - Edges: Hyperlinks



# Web as a Directed Graph



# How Can We Organize the Web?

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


- **Second try**: Web search
  - Find relevant Web pages automatically
  - (c.f.) Information retrieval
    - Find relevant documents in a small and trusted set
    - (ex) Newspaper articles, patents, etc.

- However, the Web is ***huge***, full of ***untrusted*** documents, random things, Web spam, etc.

# Web Search: Two Challenges

(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"?

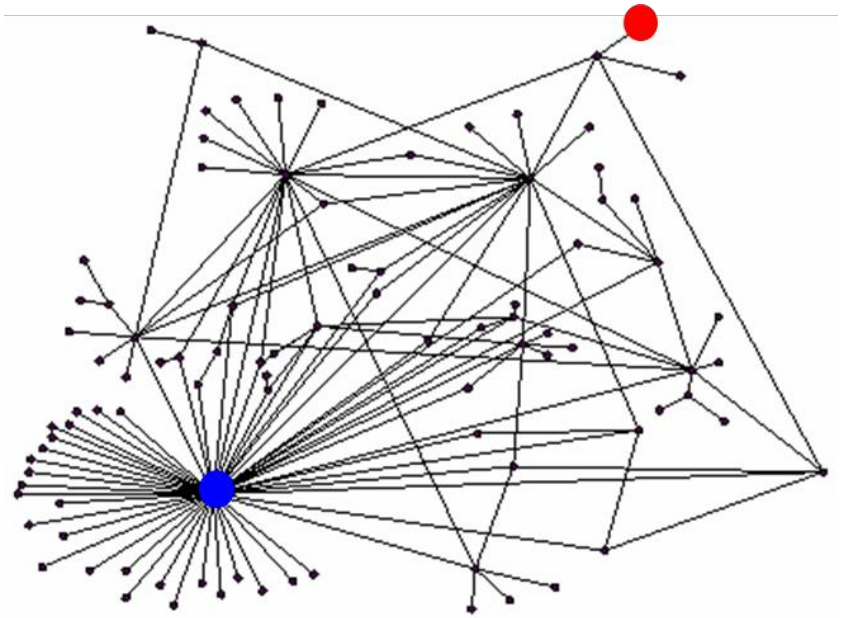
- No single right answer
- Trick: Pages that actually know about newspapers might all be pointing to many newspapers

# Ranking Nodes on the Graph

- All web pages are not equally "***important***"
  - (ex) www.joe-schmoe.com (1 in-link) vs. www.stanford.edu (23,400 in-links)

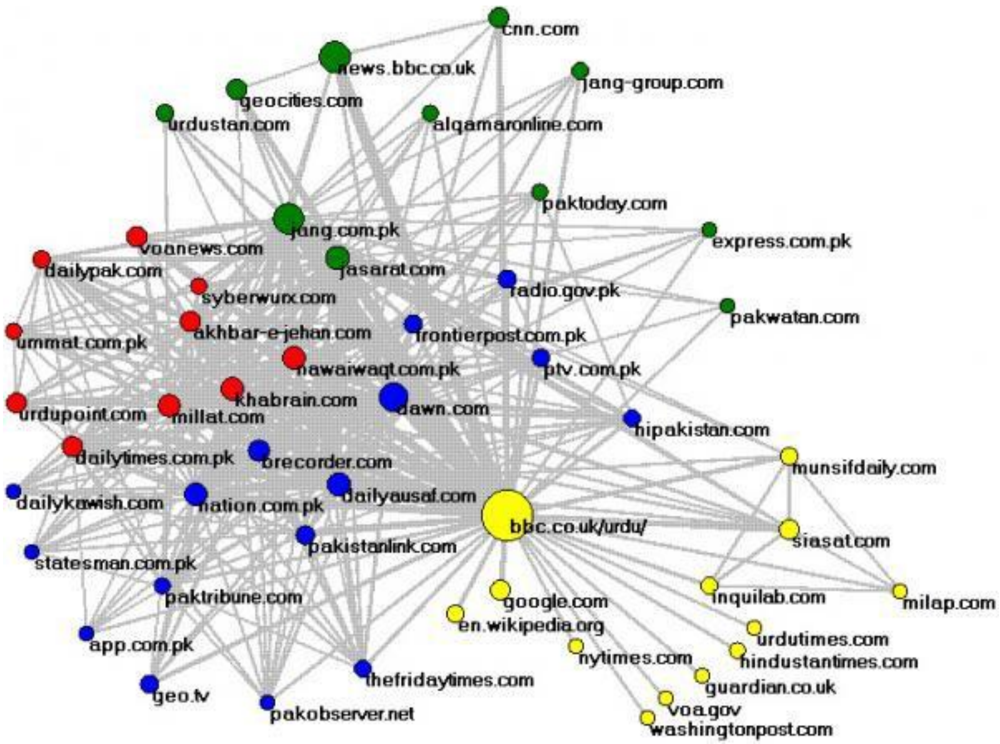
- There is large diversity in the web-graph node connectivity

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



# Link Analysis Algorithms

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



# PageRank

# Web Search

- ***Efficient*** and ***accurate*** Web search has changed our lives greatly
  - Through search engines such as Google
- Google "PageRank"
  - The first able to defeat ***spammers*** who had made search almost useless
  - We shall explain what it is and how it is computed efficiently
- Variations on PageRank
  - **TrustRank**: prevents spammers attacking PageRank (called link spam)
  - **Topic-sensitive PageRank**: weights Web pages based on their topic
  - **HITS**: "hubs and authorities" approach to evaluating Web pages

# Early Search Engines Before Google

- Crawl the Web and list the terms found in each page
- Construct an inverted index
  - A data structure used for finding all the places where a given term occurs
- When a search query (list of terms) is issued
  - Use the inverted index to retrieve the pages with those terms
  - ***Rank*** them in a way that reflects the use of the terms within the page
- Factors for page ranking
  - The number of occurrences of the term in the page
  - The place of occurrence of the term in the page (header? body?)

# Term Spam

- As people began to use search engines, unethical people tried to ***fool*** search engines into leading people to their page

- Examples
  – Add a term to your page thousands of times
    - A search engine would think your page is very important about the term
    - Thus, the search engine would list your page first
  – Make them the same color as background to hide their occurrences

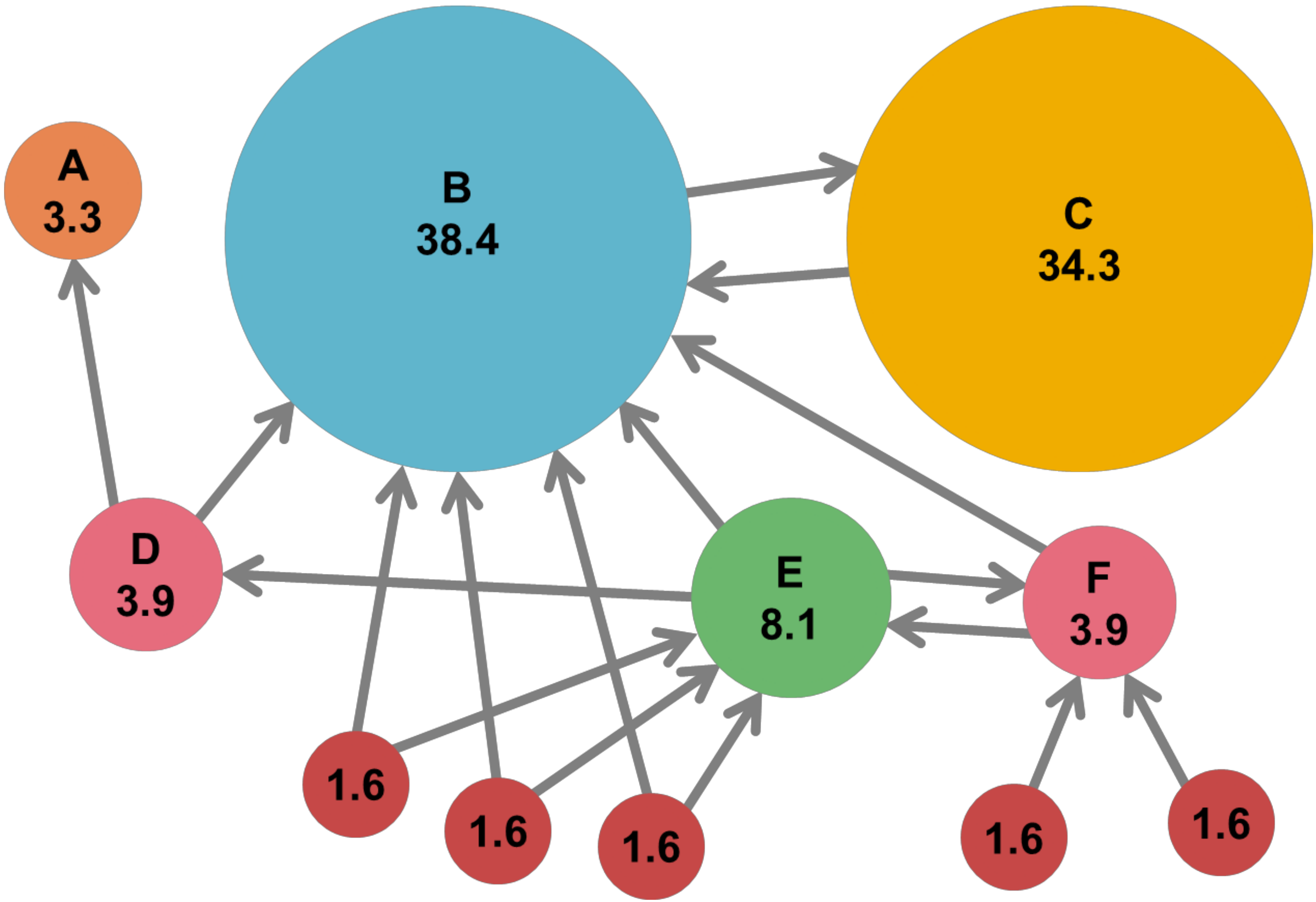
- ***Term spam***
  – Techniques for fooling search engines into believing your page is about something is not

# Google's Innovations to Combat Term Spam

1. PageRank was used to simulate Web surfers
   - They start at a random page
   - They follow randomly chosen outlinks from their current page
   - They would tend to congregate if this process iterates many time
   - Pages with a large number of surfers are considered more "***important***"
   - Google prefers important pages when deciding which pages show first

2. The content of a page was judged not only by the terms appearing on that page, but by the terms used in or near ***the links to that page***
   - Note that it is ***not*** easy for a spammer to add false terms to a page they do not control

# Example: PageRank Scores



# Why The Two Techniques Work? (1/2)

- Google believes what other pages say about him, over what he says about himself
  - While a spammer can still add a false term to his page, the above fact would negate the use of false terms
- What if the spammer creates many pages of his own, and links to his page with a link with the false term?
  - But those pages would not be given much importance by PageRank, since other pages would not link to them
  - Therefore, he still would not be able to fool Google into thinking this page is important

# Why The Two Techniques Work? (2/2)

- Why should simulation of random surfers allow us to approximate the intuitive notion of the "***importance***" of pages?

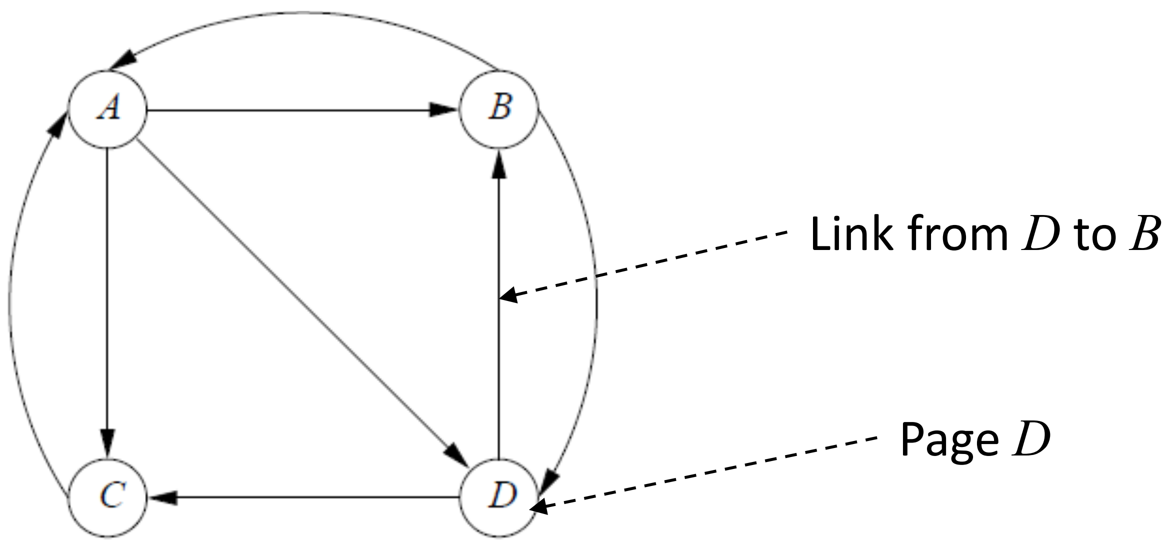
- Two related motivations
  - Users of the Web "vote with their feet"
    - They tend to place links to pages they think are good or useful pages to look at, rather than bad or useless pages
  - The behavior of a random surfer indicates which pages users of the Web are likely to visit
    - Users are more likely to visit useful pages than useless pages

- But regardless of the reason, the PageRank measure has been proved ***empirically*** to work!

# Definition of PageRank

- PageRank
  - A function that assigns a real number to each page in the Web
  - The higher the PageRank of a page, the more "***important***" it is

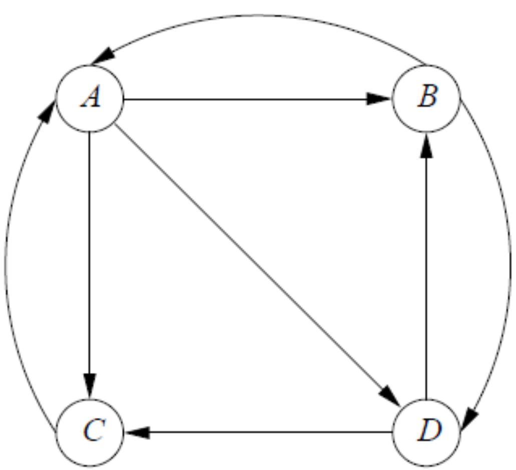
- PageRank represents the Web as a ***direct graph***
  - Pages are the nodes
  - There is an arc from page $p_1$ to page $p_2$ if there are one or more links from $p_1$ to $p_2$



# Transition Matrix of the Web

- Describes what happens to random surfers after one step

- The transition matrix $M$ is an $n \times n$ matrix
  - $n$: the number of pages
  - The element $m_{ij}$ in row $i$ and column $j$
    - $1/k$, if page $j$ has $k$ arcs out, and one of them is to page $i$
    - $0$, otherwise
    - The ***probability*** that a surfer at page $i$ will move to page $j$ at the next step




$$M = \begin{array}{cc} \begin{array}{cccc} \boldsymbol{A} & \boldsymbol{B} & \boldsymbol{C} & \boldsymbol{D} \end{array} & \\ \begin{bmatrix} 0 & 1/2 & 1 & 0 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 1/2 & 0 & 0 \end{bmatrix} & \begin{array}{c} \boldsymbol{A} \\ \boldsymbol{B} \\ \boldsymbol{C} \\ \boldsymbol{D} \end{array} \end{array}$$

# The Probability Distribution Vector

- The probability distribution vector $\mathbf{v}$
  - whose $j$th component is the probability that the surfer is at page $j$
  - $v_j$ is the (idealized) ***PageRank*** function of page $j$
  - Initially, each component has an equal value of $1/n$

- The distribution vector $\mathbf{v}$ after one step
  - $\mathbf{v} \leftarrow M\mathbf{v}$
    - Because the probability that a random surfer will be at page $i$ at the next step is $\sum_j m_{ij} v_j$

- The distribution vector $\mathbf{v}$ after two steps
  - $\mathbf{v} \leftarrow M(M\mathbf{v}) = M^2\mathbf{v}$

# A Limiting Distribution of $\mathbf{v}$

- This sort of behavior is an example of Markov process

- It is known that the distribution of the surfer approaches a ***limiting distribution*** $\mathbf{v}$ that satisfies

$$\mathbf{v} = M\mathbf{v}$$

provided two conditions are met:

  - The graph is ***strongly connected***
    - That is, it is possible to get from any node to any other node
  - There are no ***dead ends***
    - Nodes that have no arc out

# Computing a Limiting Distribution of $\mathbf{v}$

- The limit is reached when multiplying $\mathbf{v}$ by $M$ another time does not change $\mathbf{v}$
  - i.e., $\mathbf{v} = M\mathbf{v}$

- We can compute the limiting distribution $\mathbf{v}$ by
  - Starting with the initial vector $\mathbf{v_0}$ and
  - Multiplying by $M$ some number of times, until the vector we get shows little change at each round
  - (ex) $\mathbf{v_1} = M\mathbf{v_0} \rightarrow \mathbf{v_2} = M\mathbf{v_1} \rightarrow \mathbf{v_3} = M\mathbf{v_2} \rightarrow$ … until $\mathbf{v}_{i+1} \cong \mathbf{v}_i$

- In practice, for the Web itself, 50-75 iterations are sufficient to ***converge*** to within the error limits of double-precision arithmetic

# Example

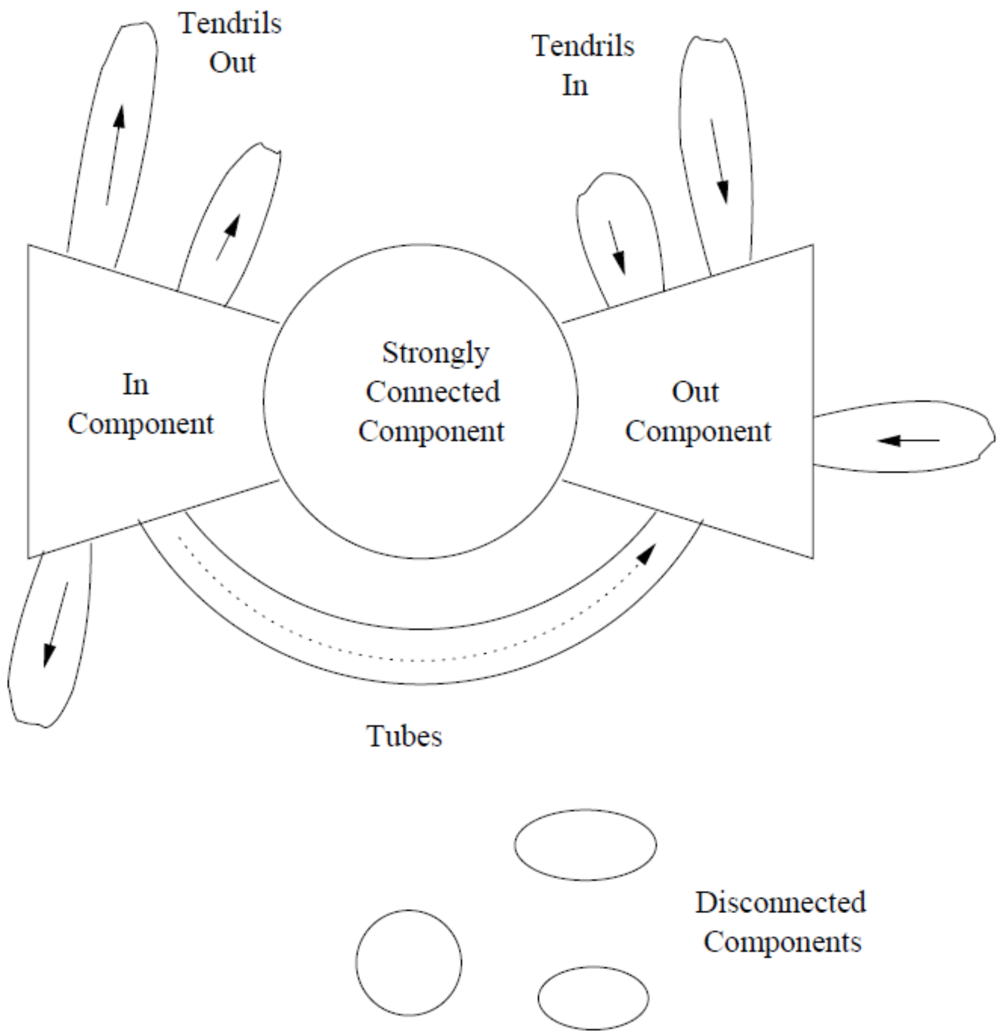
- Consider the matrix $M$ in the previous example
  - Because there are 4 nodes, the initial vector $\mathbf{v_0} = [1/4, 1/4, 1/4, 1/4]^T$
- The sequence of approximations to the limit that we get by multiplying at each step by $M$

$$\begin{bmatrix} 1/4 \\ 1/4 \\ 1/4 \\ 1/4 \end{bmatrix}, \begin{bmatrix} 9/24 \\ 5/24 \\ 5/24 \\ 5/24 \end{bmatrix}, \begin{bmatrix} 15/48 \\ 11/48 \\ 11/48 \\ 11/48 \end{bmatrix}, \begin{bmatrix} 11/32 \\ 7/32 \\ 7/32 \\ 7/32 \end{bmatrix}, \ldots, \begin{bmatrix} 3/9 \\ 2/9 \\ 2/9 \\ 2/9 \end{bmatrix}$$

  - The probability that a surfer being at $A$ is larger than that for $B$, $C$, $D$
  - Therefore, $A$ is a more ***important*** page than $B$, $C$, $D$
- In the real Web, the true probability of being at a node like www.amazon.com is orders of magnitude greater than the probability of typical nodes

# Structure of the Web

- The "bowtie" picture of the Web
  - In practice, the Web is ***not*** strongly connected



# Large Components

- A large strongly connected component (SCC)
- In-component
  - Pages that could reach the SCC by following links, but not reachable from the SCC
- Out-component
  - Pages reachable from the SCC but unable to reach the SCC
- Tendrils
  - (Type 1) Pages reachable from the in-component but not able to reach the in-component
  - (Type 2) Pages that can reach the out-component but not reachable from the out-component

# Small Components

- Tubes
  - Pages reachable from the in-component and able to reach the out-component, but unable to the SCC or be reached from the SCC

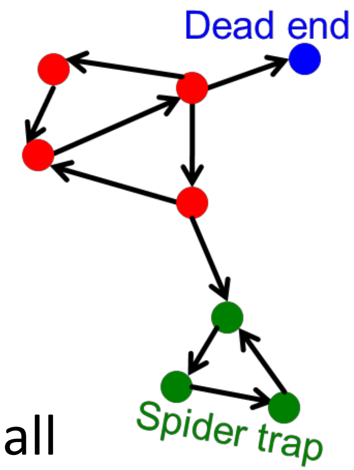
- Isolated component
  - Pages unreachable from the large components (the SCC, in- and out-components) and unable to reach those components

# Problems with the Structure of the Web

- Several of these structures ***violate*** the assumption needed for the Markov-process iteration to converge to a limit

- Example
  - When a random surfer enters either the out-component or a tendril off the in-component, they can ***never*** leave
  - Thus, ***no*** page in the SCC or in-component winds up with any probability of a surfer being there
  - Consequently, we conclude ***falsely*** that nothing in the SCC or in-component is of any importance

# How PageRank Prevent Such Anomalies




- There are really ***two problems*** we need to avoid

① Dead end

- A page that has no links out
- No page that can reach a dead end can have any PageRank at all

② Spider trap

- A group of pages that all have outlinks but never link to any other pages

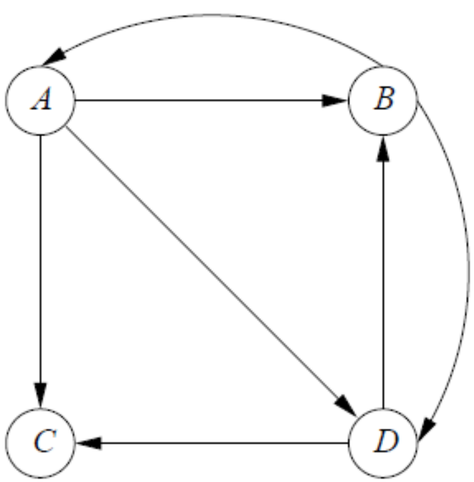
✓ Both these problems are solved by a method called "***taxation***"

- A random surfer has a finite probability of leaving the Web at any step
- New surfers are started at each page

# Avoiding Dead Ends (1/2)

- If we allow dead ends, then some of the columns of the transition matrix will sum to $0$ rather than $1$
  - Consequently, some or all of the components of $M^i\mathbf{v}$ go to $0$
  - That is, importance "drains out" of the Web
  - We get no information about the relative importance of pages

- Example
  - Consider the following transition matrix, where $C$ is a dead end
    - The sum of the third column, for $C$, is $0$, not $1$




$$
\begin{array}{cc}
 & \begin{array}{cccc} \boldsymbol{A} & \boldsymbol{B} & \boldsymbol{C} & \boldsymbol{D} \end{array} \\
M = & \left[\begin{array}{cccc} 0 & 1/2 & 0 & 0 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 1/2 & 0 & 0 \end{array}\right] \begin{array}{c} \boldsymbol{A} \\ \boldsymbol{B} \\ \boldsymbol{C} \\ \boldsymbol{D} \end{array}
\end{array}
$$

# Avoiding Dead Ends (2/2)

- Example (cont'd)
  - The sequence of vectors that result by starting the vector with each component $1/4$, and repeatedly multiplying the vector by $M$

$$M = \begin{bmatrix} 0 & 1/2 & 0 & 0 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 1/2 & 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 1/4 \\ 1/4 \\ 1/4 \\ 1/4 \end{bmatrix}, \begin{bmatrix} 3/24 \\ 5/24 \\ 5/24 \\ 5/24 \end{bmatrix}, \begin{bmatrix} 5/48 \\ 7/48 \\ 7/48 \\ 7/48 \end{bmatrix}, \begin{bmatrix} 21/288 \\ 31/288 \\ 31/288 \\ 31/288 \end{bmatrix}, \ldots, \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

  - As we see, the probability of a surfer being anywhere goes to $0$, as the number of steps increase

# Two Approaches to Dealing with Dead Ends

1. Recursive deletion of dead ends
   - Drop the dead ends from the graph, and also drop their incoming arcs
   - Doing so may create more dead ends, which also have to be dropped, recursively
   - However, eventually we wind up with a strongly-connected component, none of whose nodes are dead ends

2. Taxation
   - Modify the process by which random surfers are assumed to move about the Web
   - This method also solves the problem of spider traps

# Recursive Deletion of Dead Ends (1/2)

1. Drop the dead ends from the graph recursively

2. Solve the remaining graph $G$
   – By whatever means are appropriate, including the taxation method

3. Restore the graph
   – But keep the PageRank values for the nodes of $G$

4. Compute the PageRank of nodes not in $G$, but with predecessors all in $G$
   – By summing, over all predecessors $p$, the PageRank of $p$ divided by the number of successors of $p$ in the full graph

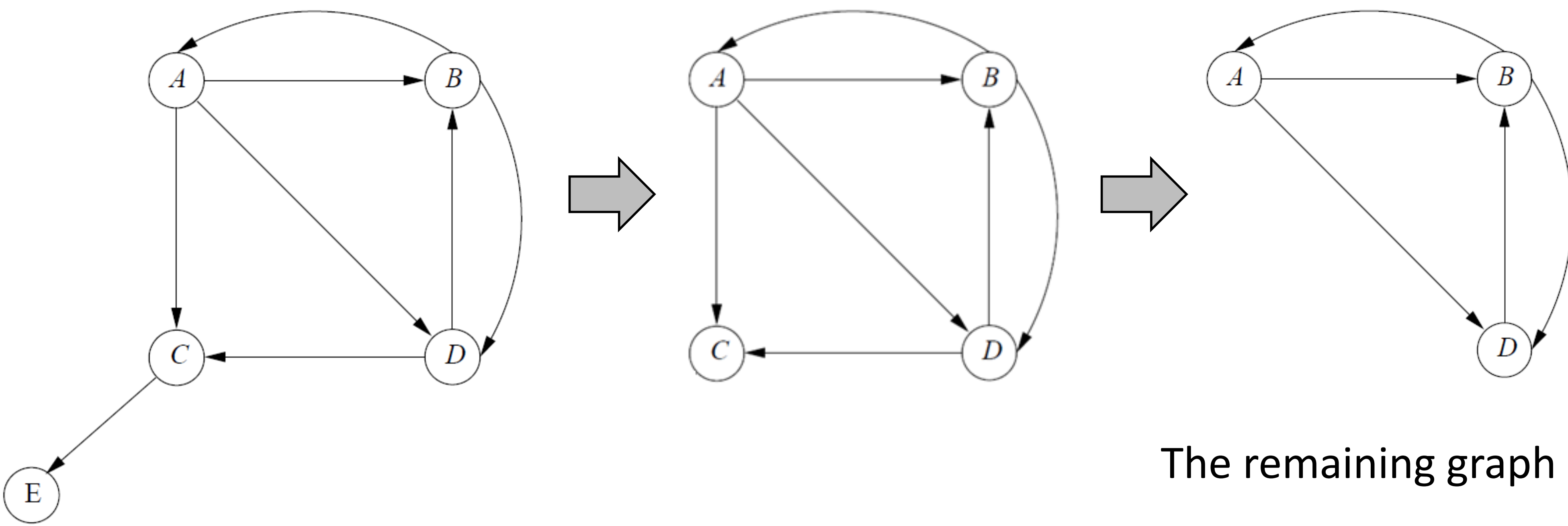
# Recursive Deletion of Dead Ends (2/2)

5. Compute the PageRank of other nodes, not in $G$, that have the PageRank of all their predecessors computed
   - By the same process

6. Eventually, all nodes outside $G$ will have their PageRank computed
   - They can surely be computed in the order ***opposite*** to that in which they where deleted

# Example (1/3)

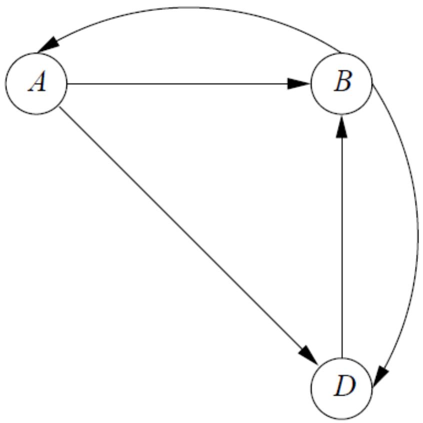
- Consider the following graph, where $E$ is a dead end
  - After removing $E$, we find that $C$ is now a dead end
  - After removing $C$, there are no more dead ends




The remaining graph $G$

# Example (2/3)

- The transition matrix $M$ for the remaining graph $G$



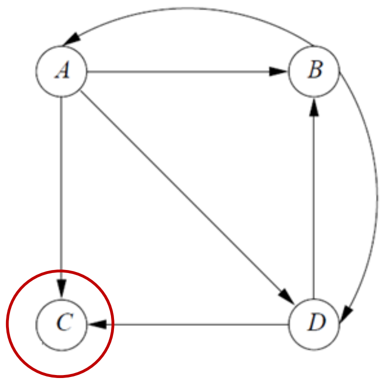

$$M = \begin{array}{c} \begin{array}{ccc} A & B & D \end{array} \\ \begin{bmatrix} 0 & 1/2 & 0 \\ 1/2 & 0 & 1 \\ 1/2 & 1/2 & 0 \end{bmatrix} \end{array} \begin{array}{c} \\ A \\ B \\ D \end{array}$$

- We get the PageRanks for this matrix
  - By starting with a vector with all components equals to $1/3$, and repeatedly multiplying by $M$

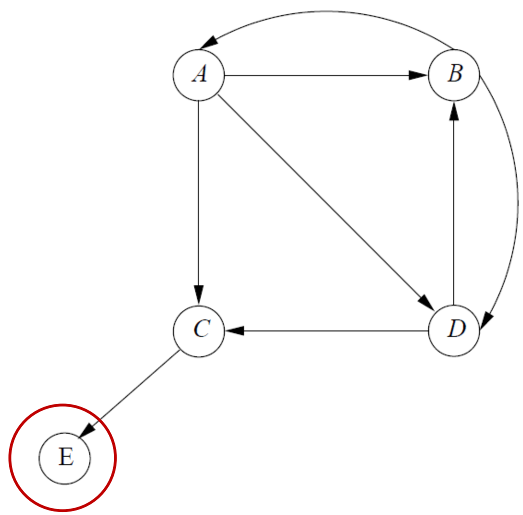
$$\begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix}, \begin{bmatrix} 1/6 \\ 3/6 \\ 2/6 \end{bmatrix}, \begin{bmatrix} 3/12 \\ 5/12 \\ 4/12 \end{bmatrix}, \begin{bmatrix} 5/24 \\ 11/24 \\ 8/24 \end{bmatrix}, \ldots, \boxed{\begin{bmatrix} 2/9 \\ 4/9 \\ 3/9 \end{bmatrix}} \begin{array}{c} A \\ B \\ D \end{array}$$

# Example (3/3)

- Now we compute PageRank for $C$
  – Because all its predecessors (i.e., $A$ and $B$) have PageRanks computed
  – The PageRank for C $= 1/3 \times 2/9 + 1/2 \times 3/9 = 13/54$



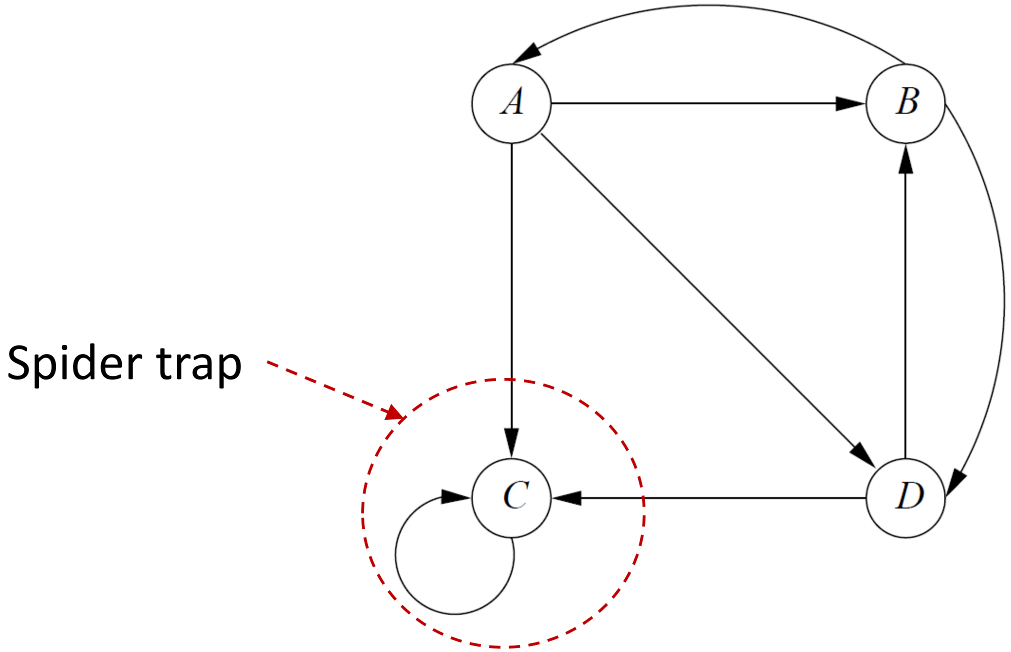

- Finally we compute PageRank for $E$
  – Because all its predecessors (i.e., $C$) have PageRanks computed
  – The PageRank for $E = 13/54$




✓ Note that the sums of the PageRank ***exceed*** $1$
  – They no longer represent the distribution of a random surfer
  – Yet they represent decent estimates of the relative importance of the pages

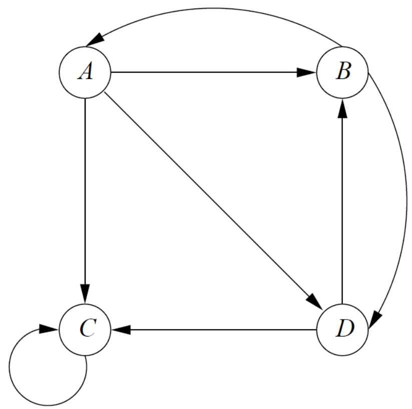
# Spider Trap

- A set of nodes with no dead ends but no arcs out
- Can appear ***intentionally*** or unintentionally on the Web
- Causes the PageRank calculation to place all the PageRank within the spider trap



# Example

- Consider the following graph, where $C$ is a spider trap of $1$ node, and its transition matrix $M$




$$M = \begin{bmatrix} 0 & 1/2 & 0 & 0 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 0 & 1 & 1/2 \\ 1/3 & 1/2 & 0 & 0 \end{bmatrix}$$

- The usual iteration to compute the PageRank

$$\begin{bmatrix} 1/4 \\ 1/4 \\ 1/4 \\ 1/4 \end{bmatrix}, \begin{bmatrix} 3/24 \\ 5/24 \\ 11/24 \\ 5/24 \end{bmatrix}, \begin{bmatrix} 5/48 \\ 7/48 \\ 29/48 \\ 7/48 \end{bmatrix}, \begin{bmatrix} 21/288 \\ 31/288 \\ 205/288 \\ 31/288 \end{bmatrix}, \ldots, \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

  - As predicted, ***all the PageRank is at $C$***
  - Since once there a random surfer can never leave

# Taxation

- To avoid this problem, we allow each random surfer a small probability of ***teleporting*** to a random page
  - Rather than following an out-link from their current page

- The calculation of PageRank is modified as follows:

$$\mathbf{v}' = \beta M\mathbf{v} + (1-\beta)\mathbf{e}/n$$

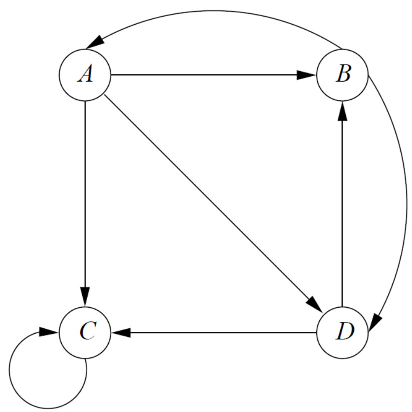
  - $\beta$ : a chosen constant (usually in the range $0.8$ to $0.9$)
  - $\mathbf{e}$ : a vector of all $1$'s with the appropriate number of components
  - $n$ : the number of nodes in the Web graph
  - $\beta M\mathbf{v}$ : represents the case where, with probability $\beta$, the random surfer follows an out-link from their present page
  - $(1-\beta)\mathbf{e}/n$ : the introduction, with probability $1-\beta$, of a new random surfer at a random page

# Taxation and Dead Ends

- If the graph has no dead ends
  - The surfer decides either to follow a link or teleport to a random page
- If the graph has dead ends
  - There is a third possibility that the surfer goes nowhere
  - Due to the term $(1-\beta)\mathbf{e}/n$, there will always be some fraction of a surfer operating on the Web
  - Thus, when there are dead ends, the sum of the components of $\mathbf{v}$ may less than $1$, but it will ***never*** reach $0$

# Example (1/2)

- Suppose we apply the new approach to the following graph




$$M = \begin{bmatrix} 0 & 1/2 & 0 & 0 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 0 & 1 & 1/2 \\ 1/3 & 1/2 & 0 & 0 \end{bmatrix}$$

- If we use $\beta = 0.8$, then the equation for the iteration becomes

$$\mathbf{v}' = \begin{bmatrix} 0 & 2/5 & 0 & 0 \\ 4/15 & 0 & 0 & 2/5 \\ 4/15 & 0 & 4/5 & 2/5 \\ 4/15 & 2/5 & 0 & 0 \end{bmatrix} \mathbf{v} + \begin{bmatrix} 1/20 \\ 1/20 \\ 1/20 \\ 1/20 \end{bmatrix}$$

# Example (2/2)

- Here are the first few iterations:

$$\begin{bmatrix} 1/4 \\ 1/4 \\ 1/4 \\ 1/4 \end{bmatrix}, \begin{bmatrix} 9/60 \\ 13/60 \\ 25/60 \\ 13/60 \end{bmatrix}, \begin{bmatrix} 41/300 \\ 53/300 \\ 153/300 \\ 53/300 \end{bmatrix}, \begin{bmatrix} 543/4500 \\ 707/4500 \\ 2543/4500 \\ 707/4500 \end{bmatrix}, \ldots, \begin{bmatrix} 15/148 \\ 19/148 \\ \boxed{95/148} \\ 19/148 \end{bmatrix}$$

- By being a spider trap, $C$ has managed to get more than half of the PageRank for itself

- However, the effect has been ***limited***, and each of the nodes gets some of the PageRank

# Using PageRank in a Search Engine (1/2)

- After a search engine has crawled the portion of the Web, the ***PageRank vector*** for that portion is calculated
- Each search engine has a ***secrete formula*** that decides the order in which to show pages to the user in response to a search query consisting of one or more search terms (words)
- Google is said to use over 250 different properties of pages, from which a linear order of pages is decided

# Using PageRank in a Search Engine (2/2)

- First, in order to be considered for the ranking at all, a page has to have at least one of the search terms in the query
  - Normally, a page has very little chance of being in the top ten unless all the search terms are present

- Among the qualified pages, a score is computed for each
  - An important component of this score is the ***PageRank*** of the page
  - Other components include the presence or absence of search terms in prominent places, such as headers or the links to the page itself

# Efficient Computation of PageRank

# Efficient Computation of PageRank

- To compute the PageRank, we have to perform a matrix-vector multiplication on the order of ***50*** times
  - Until the vector is close to unchanged at one iteration

- To a first approximation, the MapReduce method given in Chapter 2 is suitable

- However, we must deal with two issues:
  1. The transition matrix of the Web $M$ is very ***sparse***
     - Representing it by all its elements is highly inefficient
  2. We may ***not*** be using MapReduce, or may wish to use a ***combiner***
     - In this case, the striping approach discussed in Chapter 2 is not sufficient to avoid heavy use of disk (thrashing)

# Representing Transition Matrices

- The transition matrix is very ***sparse***
  - Since the average Web page has about 10 out-links
  - (ex) If we are analyzing a graph of 10 billion pages, then only one in a billion entries is not 0

- The proper way to represent a sparse matrix
  - List the locations of the ***nonzero*** entries and their values
  - Thus, the space needed is linear in the number of nonzero entries, rather than quadratic in the side of the matrix
    - (ex) 16 bytes per nonzero entry (4-byte integers for coordinates of an elements and an 8 byte double-precision number for the value)

# Further Compression for a Transition Matrix

- If we list the nonzero entries by columns, then we know what the value of each nonzero entry is
  - It is 1 divided by the out-degree of the page
- We can thus represent a column by
  - One integer for the out-degree
  - One integer per nonzero entry in that column (i.e., the row number)
- Example

$$M = \begin{bmatrix} 0 & 1/2 & 1 & 0 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 1/2 & 0 & 0 \end{bmatrix}$$

| Source | Degree | Destinations |
|---|---|---|
| $A$ | 3 | $B$, $C$, $D$ |
| $B$ | 2 | $A$, $D$ |
| $C$ | 1 | $A$ |
| $D$ | 2 | $B$, $C$ |

A compact representation of $M$

# PageRank Iteration Using MapReduce

- Recall that each iteration of the PageRank algorithm computes

$$\mathbf{v}' = \beta M\mathbf{v} + (1 - \beta)\mathbf{e}/n$$

- If each Map task can store the full vector $\mathbf{v}$ in memory and also have room in main memory for the result vector $\mathbf{v}'$
  – Then there is little more here than a matrix-vector multiplication
  – Additional steps: to multiply each component of $M\mathbf{v}$ by constant $\beta$ and to add $(1 - \beta)/n$ to each component

- However, it is likely that $\mathbf{v}$ is much too large to fit in main memory
  – Then, we can use the striping method described in Chapter 2 to execute the MapReduce process efficiently
  – That is, we break $M$ into vertical stripes and break $\mathbf{v}$ into corresponding horizontal stripes

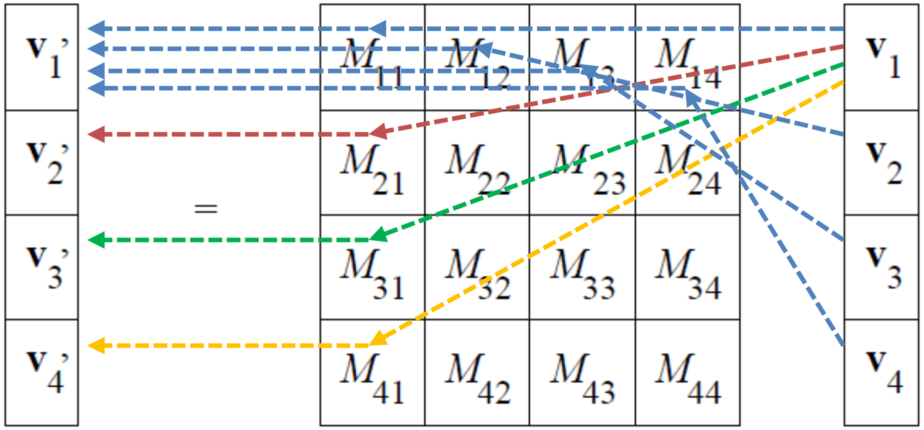
# Use of Combiners

- The previous method might ***not*** be adequate for two reasons:
  1. We might wish to add terms for $\mathbf{v}_i$' at the Map tasks
     - $\mathbf{v}_i$' : the $i$th component of the result vector $\mathbf{v}$
     - The Reduce function simply adds terms with a common key
     - Recall that for a MapReduce implementation of matrix-vector multiplication, the key is the value of $i$ for which a term $m_{ij}\mathbf{v}_j$ is intended
  2. We might not be using MapReduce at all

- We are trying to implement a ***combiner*** in conjunction with a Map task
  - The second case uses essentially the same idea

# An Alternative Strategy (1/2)

- Partition the matrix into $k^2$ blocks
  - The vectors are still partitioned into $k$ stripes
  - (ex) partitioning the matrix with $k = 4$




- We use $k^2$ Map tasks
  - Each task gets one square of $M$, say $M_{ij}$, and one stripe of $\mathbf{v}$, $\mathbf{v}_j$
  - Each stripe of $\mathbf{v}$ is sent to $k$ different Map tasks
    - $\mathbf{v}_j \rightarrow M_{1j}, M_{2j}, \ldots, M_{kj}$

# An Alternative Strategy (2/2)

- The transmission cost
  - Each $\mathbf{v}_j$ is transmitted over the network $k$ times
  - Each $M_{ij}$ is transmitted only once
  - Since the size of $M$ is expected to be several times the size of $\mathbf{v}$, the transmission cost is not too much greater than the minimum possible
  - Because we are doing considerable combining at the Map tasks, we save as data is passed from the Map tasks to the Reduce tasks

- The advantage of this approach
  - We can keep both the $j$th stripe of $\mathbf{v}$ and the $i$th stripe of $\mathbf{v}$' in main memory as we process $M_{ij}$
  - Thus, we don't need to access disk frequently to bring them into main memory (***thrashing***)

# Representing Blocks of the Transition Matrix

- Since we are representing transition matrices in the special way described previously, we need to consider how the blocks of a transition matrix are represented

- Example
  - Suppose the following matrix is partitioned into blocks with $k = 2$

$$M = \begin{bmatrix} 0 & 1/2 & 1 & 0 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 1/2 & 0 & 0 \end{bmatrix} \Rightarrow M = \left[\begin{array}{cc|cc} 0 & 1/2 & 1 & 0 \\ 1/3 & 0 & 0 & 1/2 \\ \hline 1/3 & 0 & 0 & 1/2 \\ 1/3 & 1/2 & 0 & 0 \end{array}\right]$$

| Source | Degree | Destinations |
|---|---|---|
| $A$ | 3 | $B$, $C$, $D$ |
| $B$ | 2 | $A$, $D$ |
| $C$ | 1 | $A$ |
| $D$ | 2 | $B$, $C$ |

$\Rightarrow$ ?

# Representing Blocks of the Transition Matrix

- Example (cont'd)

$$
\begin{array}{ccccccc}
 & A & B & C & D & \\
M = & \left[\begin{array}{cc|cc} 0 & 1/2 & 1 & 0 \\ 1/3 & 0 & 0 & 1/2 \\ \hline 1/3 & 0 & 0 & 1/2 \\ 1/3 & 1/2 & 0 & 0 \end{array}\right] & \begin{array}{c} A \\ B \\ C \\ D \end{array}
\end{array}
$$

| Source | Degree | Destinations |
|---|---|---|
| $A$ | 3 | $B$ |
| $B$ | 2 | $A$ |

Representation of $M_{11}$
(connecting $A$ and $B$ to $A$ and $B$)

| Source | Degree | Destinations |
|---|---|---|
| $C$ | 1 | $A$ |
| $D$ | 2 | $B$ |

Representation of $M_{12}$
(connecting $C$ and $D$ to $A$ and $B$)

We must repeat

| Source | Degree | Destinations |
|---|---|---|
| $A$ | 3 | $C$, $D$ |
| $B$ | 2 | $D$ |

Representation of $M_{21}$
(connecting $A$ and $B$ to $C$ and $D$)

| Source | Degree | Destinations |
|---|---|---|
| $D$ | 2 | $C$ |

(we can avoid the entry for $C$)

Representation of $M_{22}$
(connecting $C$ and $D$ to $C$ and $D$)

# Other Efficient Approaches to PageRank Iteration

- The $k^2$ partitioning algorithm is not the only option
  – There are several other approaches that use ***fewer*** processors
- These algorithms share the following good property
  – The matrix $M$ is read only once, although the vector $\mathbf{v}$ is read $k$ times
  – $k$ is chosen so that $1/k$th of $\mathbf{v}$ and $\mathbf{v}'$ can be held in main memory
- Recall that the $k^2$ partitioning algorithm uses $k^2$ processors, assuming all Map tasks are executed in parallel at different processors

# Other Efficient Approaches to PageRank Iteration

- We can assign all the blocks in one row to a single Map task
  - Thus the number of Map tasks is reduced to $k$

$$\begin{bmatrix} \mathbf{v}'_1 \\ \mathbf{v}'_2 \\ \mathbf{v}'_3 \\ \mathbf{v}'_4 \end{bmatrix} = \begin{bmatrix} M_{11} & M_{12} & M_{13} & M_{14} \\ M_{21} & M_{22} & M_{23} & M_{24} \\ M_{31} & M_{32} & M_{33} & M_{34} \\ M_{41} & M_{42} & M_{43} & M_{44} \end{bmatrix} \begin{bmatrix} \mathbf{v}_1 \\ \mathbf{v}_2 \\ \mathbf{v}_3 \\ \mathbf{v}_4 \end{bmatrix}$$

Assigned to
a single Map task

  - We can read the blocks one-at-a-time, so the matrix does not consume a significant amount of main memory
  - At the same time that we read $M_{ij}$, we must read the vector stripe $\mathbf{v}_j$
  - As a result, each of the $k$ Map tasks reads the entire vector $\mathbf{v}$
  - **Advantage:** each Map task can combine ***all*** the terms for the portion $\mathbf{v}_i'$ for which it is exclusively responsible

# Other Efficient Approaches to PageRank Iteration

- We can extend this idea to an environment where MapReduce is ***not*** used
  - Suppose we have a single processor, with $M$ and $\mathbf{v}$ stored on its disk
- We simulate each Map task one by one
  - The first task: reads $M_{11}$ through $M_{1k}$ and all of $\mathbf{v}$ to compute $\mathbf{v}_1$'
  - The second task: reads $M_{21}$ through $M_{2k}$ and all of $\mathbf{v}$ to compute $\mathbf{v}_2$'
  - …
  - The $k$th task: reads $M_{k1}$ through $M_{kk}$ and all of $\mathbf{v}$ to compute $\mathbf{v}_k$'
- We can make $k$ as small as possible
  - Subject to the constraint that there is enough main memory to store $1/k$th of $\mathbf{v}$ and $1/k$th of $\mathbf{v}$', along with as small a portion of $M$ as we can read from disk

# Some Problems with PageRank

- Measures ***generic*** popularity of a page
  - Biased against topic-specific authorities
  - **Solution**: Topic-Specific PageRank (next)

- Susceptible to ***Link spam***
  - Artificial link topographies created in order to boost page rank
  - **Solution**: TrustRank

- Uses a ***single*** measure of importance
  - Other models of importance
  - **Solution**: Hubs-and-Authorities