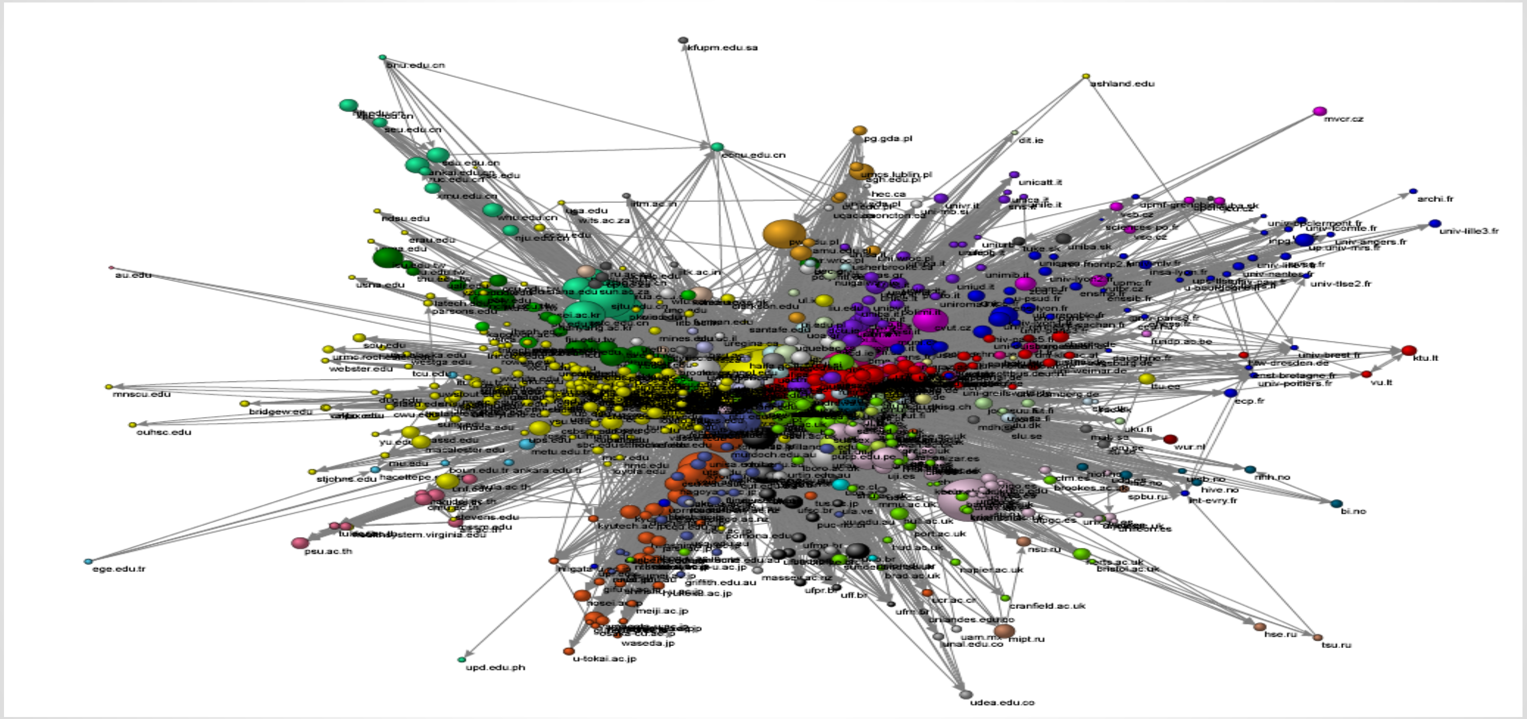
Web Graph Analysis: Bring Order to the Web



Presented by:

Sahil Mutneja(B11031) Ankit Srivastava(B11108) Sachin Tyagi(B11030)

Early Search Engines

- Worked by crawling the web and listing the terms found in each page in an **Inverted Index**.

- An **Inverted Index** is a data structure that makes it easy, given a term to find all the places where that term occurs.

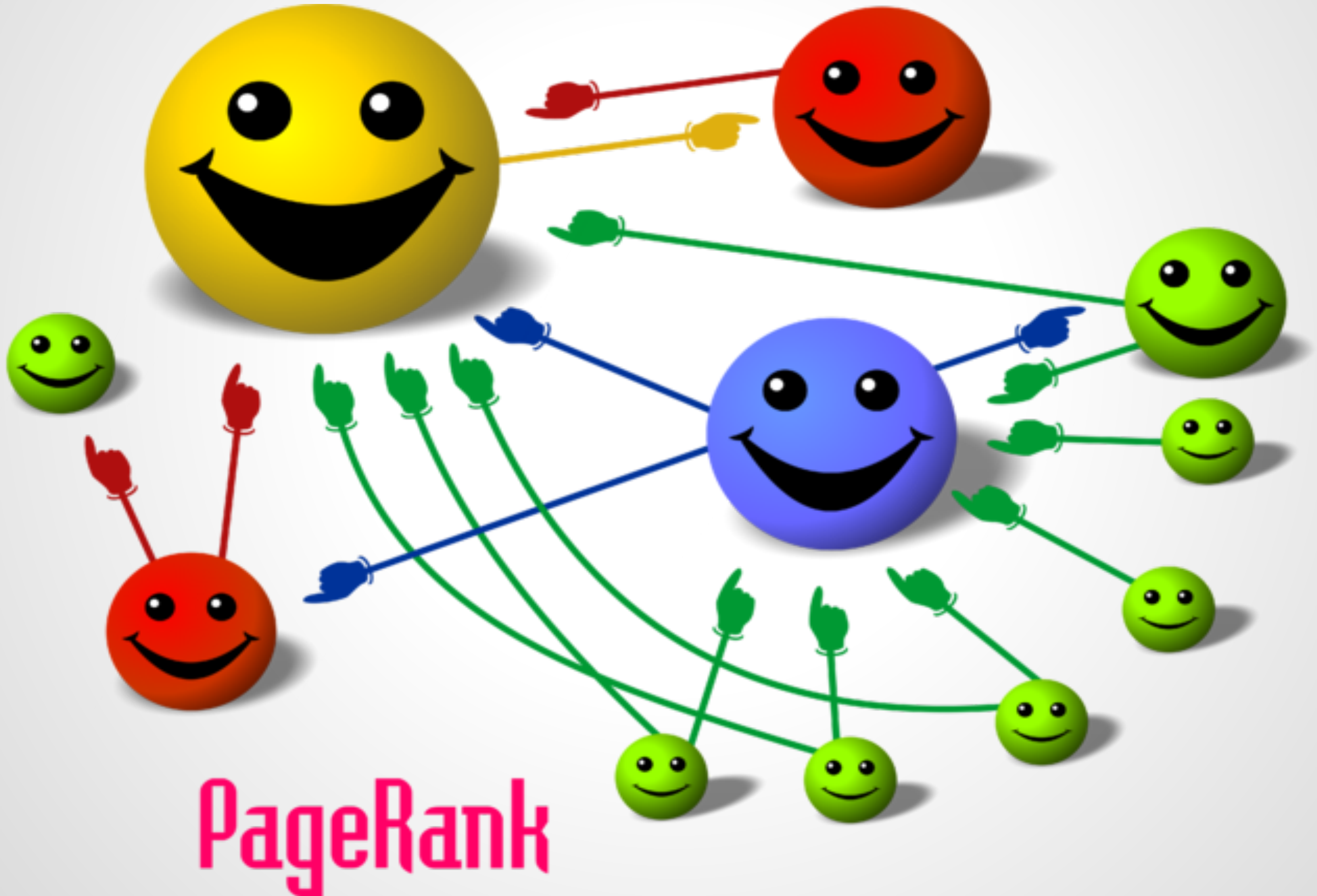
- Finally the pages related to search query were extracted from the inverted index and ranked in a way that reflected the use of the terms within the page.

Term Spam

- As search engines became popular, unethical people saw the opportunity to fool search engines into leading people to their page.

- **Example**:
  If you were selling shoes on the web, you would want people to visit your page, thus you could add a popular term like movie etc. to your page and do it thousands of time so a search engine would think that your page is terribly important about movies.
  When a user issued a search query with the term movie, the search engine will list your page first.

Diving into the Concept of PageRank

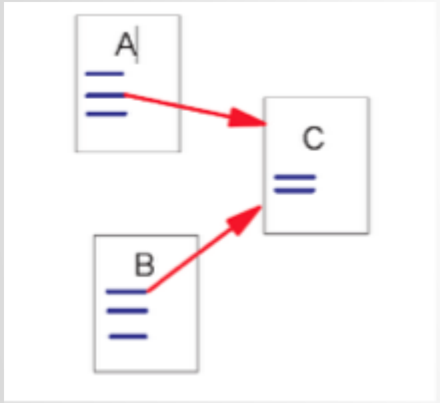


Definition of PageRank, Covering Basics

- PageRank is a function that assigns a real number to each page in the Web (or at least to that portion of the Web that has been crawled and its links discovered).
- The intent is that the higher the PageRank of a page, the more "*important*" it is.
- A method for rating the importance of web pages objectively and mechanically using the link structure of the web.

Link Structure of the Web

150 million web pages -> 1.7 billion links



Backlinks and Forward Links:

- ➔ A and B are C’s backlinks.
- ➔ C is A and B’s forward link

Intuitively a page is important if it has a lot of backlinks to it

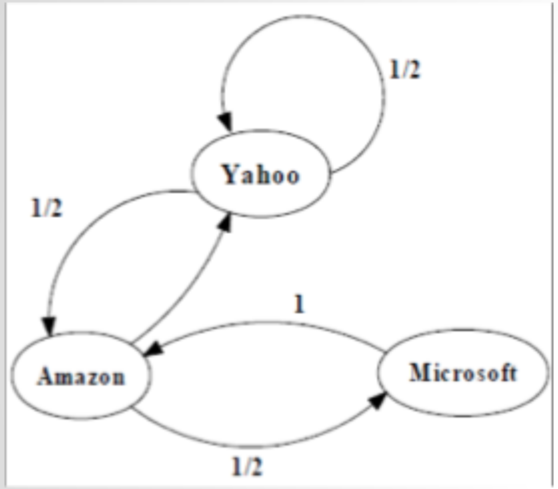
A simple version of PageRank

$$R(u) = c \sum_{v \in B_u} \frac{R(v)}{N_v}$$

u : a web page
B_u : the set of u’s backlinks
$R(v)$: PageRank of page v

N_v : the number of forward links of page v
c : a constant

An example of Simplified PageRank

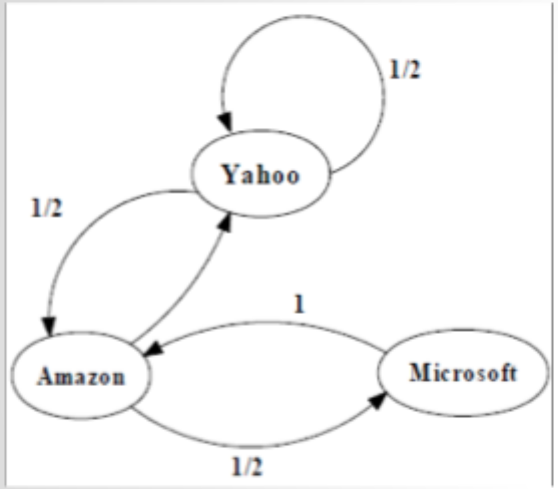


$$M = \begin{bmatrix} 1/2 & 1/2 & 0 \\ 1/2 & 0 & 1 \\ 0 & 1/2 & 0 \end{bmatrix}. \quad \begin{bmatrix} \text{yahoo} \\ \text{Amazon} \\ \text{Microsoft} \end{bmatrix} = \begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix}$$

PageRank Calculation: first iteration

$$\begin{bmatrix} 1/3 \\ 1/2 \\ 1/6 \end{bmatrix} = \begin{bmatrix} 1/2 & 1/2 & 0 \\ 1/2 & 0 & 1 \\ 0 & 1/2 & 0 \end{bmatrix} \begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix}$$

An example of Simplified PageRank(2)

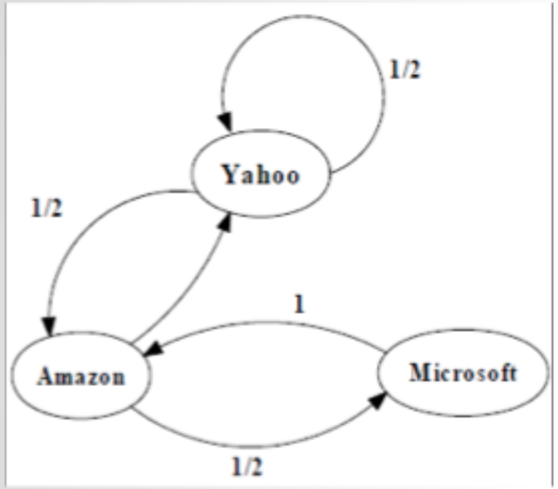


$$M = \begin{bmatrix} 1/2 & 1/2 & 0 \\ 1/2 & 0 & 1 \\ 0 & 1/2 & 0 \end{bmatrix} \qquad \begin{bmatrix} \text{yahoo} \\ \text{Amazon} \\ \text{Microsoft} \end{bmatrix} = \begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix}$$

PageRank Calculation: second iteration

$$\begin{bmatrix} 5/12 \\ 1/3 \\ 1/4 \end{bmatrix} = \begin{bmatrix} 1/2 & 1/2 & 0 \\ 1/2 & 0 & 1 \\ 0 & 1/2 & 0 \end{bmatrix} \begin{bmatrix} 1/3 \\ 1/2 \\ 1/6 \end{bmatrix}$$

An example of Simplified PageRank(3)



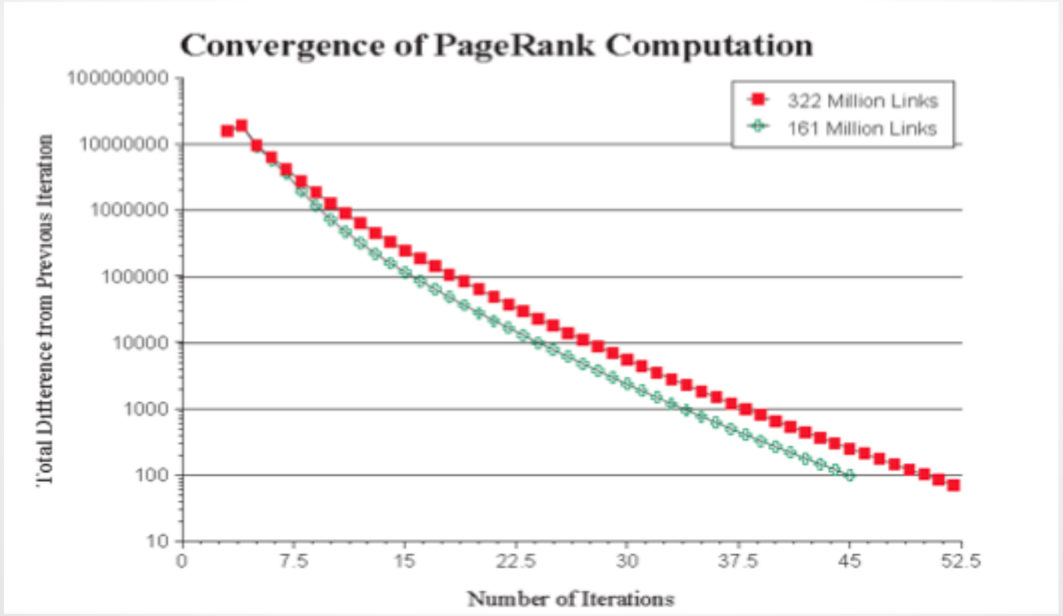
$$M = \begin{bmatrix} 1/2 & 1/2 & 0 \\ 1/2 & 0 & 1 \\ 0 & 1/2 & 0 \end{bmatrix}, \quad \begin{bmatrix} \text{yahoo} \\ \text{Amazon} \\ \text{Microsoft} \end{bmatrix} = \begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix}$$

Convergence after some iterations

$$\begin{bmatrix} 3/8 \\ 11/24 \\ 1/6 \end{bmatrix} \begin{bmatrix} 5/12 \\ 17/48 \\ 11/48 \end{bmatrix} \ldots \begin{bmatrix} 2/5 \\ 2/5 \\ 1/5 \end{bmatrix}$$

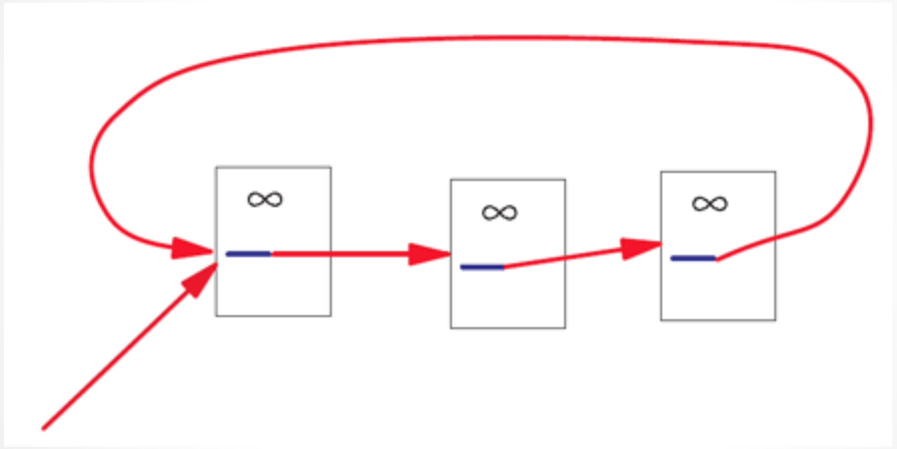
Convergence Property

- PR (322 Million Links) : 52 iterations
- PR (161 Million Links) : 45 iterations
- Scaling factor is roughly linear in *logn*



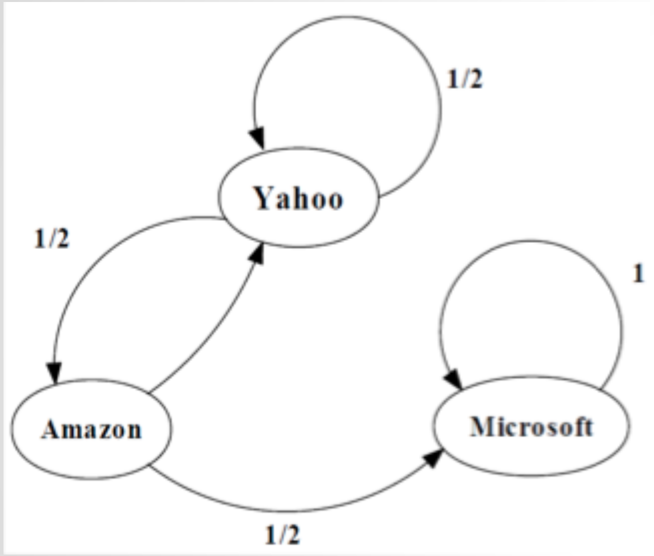
Problem with Simplified PageRank- Spider Trap

A Loop:



During each iteration the loop accumulates rank but never distributes rank to other pages. This problem is known as *spider traps.* These are the group of pages that all have outlinks but they never link to any other pages.

An example of the Problem

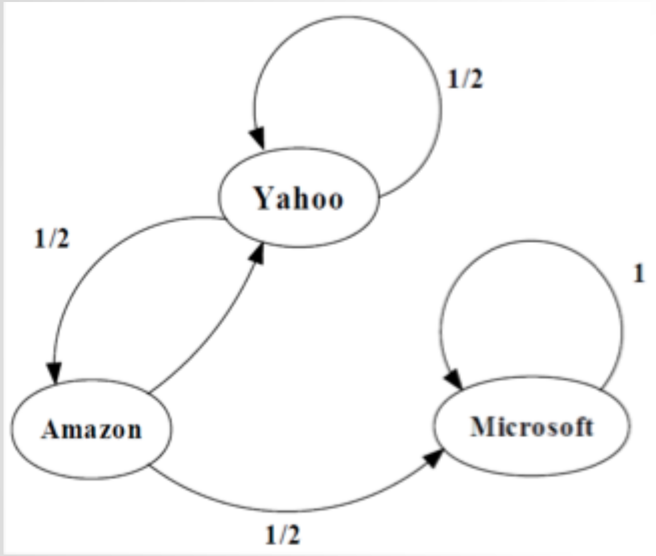


$$M = \begin{bmatrix} 1/2 & 1/2 & 0 \\ 1/2 & 0 & 0 \\ 0 & 1/2 & 1 \end{bmatrix} \quad \begin{bmatrix} \text{yahoo} \\ \text{Amazon} \\ \text{Microsoft} \end{bmatrix} = \begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix}$$

PageRank Calculation: first iteration

$$\begin{bmatrix} 1/3 \\ 1/6 \\ 1/2 \end{bmatrix} = \begin{bmatrix} 1/2 & 1/2 & 0 \\ 1/2 & 0 & 0 \\ 0 & 1/2 & 1 \end{bmatrix} \begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix}$$

An example of the Problem

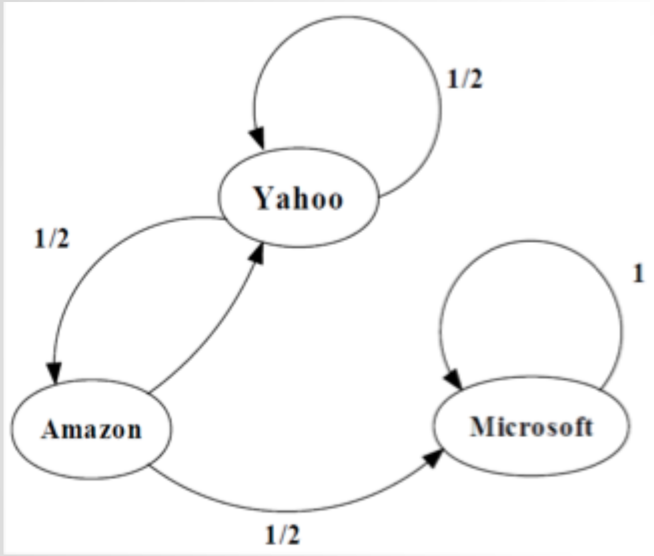


$$M = \begin{bmatrix} 1/2 & 1/2 & 0 \\ 1/2 & 0 & 0 \\ 0 & 1/2 & 1 \end{bmatrix} \quad \begin{bmatrix} \text{yahoo} \\ \text{Amazon} \\ \text{Microsoft} \end{bmatrix} = \begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix}$$

PageRank Calculation: second iteration

$$\begin{bmatrix} 1/4 \\ 1/6 \\ 7/12 \end{bmatrix} = \begin{bmatrix} 1/2 & 1/2 & 0 \\ 1/2 & 0 & 0 \\ 0 & 1/2 & 1 \end{bmatrix} \begin{bmatrix} 1/3 \\ 1/6 \\ 1/2 \end{bmatrix}$$

An example of the Problem



$$M = \begin{bmatrix} 1/2 & 1/2 & 0 \\ 1/2 & 0 & 0 \\ 0 & 1/2 & 1 \end{bmatrix} \cdot \begin{bmatrix} \text{yahoo} \\ \text{Amazon} \\ \text{Microsoft} \end{bmatrix} = \begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix}$$

Convergence after some iterations

$$\begin{bmatrix} 5/24 \\ 1/8 \\ 2/3 \end{bmatrix} \begin{bmatrix} 1/6 \\ 5/48 \\ 35/48 \end{bmatrix} \cdots \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

Problem with Simplified PageRank- Dangling Links

- Links that point to any page with no outgoing links. This page with no outgoing links is known as *dead end*.
- The probability of a surfer being anywhere goes to 0, as the number of steps increases.
- There are two approaches to deal with dead ends :
  - *Remove Dead Ends*- Recursively drop dead ends from the graph along with their incoming arcs. Eventually, we will be having a strongly connected component with no dead ends.
  - *Taxation*- Overcomes the problem of dead end and spider traps, we add another term to the basic formula which consider the probability of moving from one page to another without following any link.

Searching with PageRank

- Two search engines
  - Title Based Search Engine
  - Full Text Search Engine
- *Title Based Search Engine*
  - Searches only the titles
  - Find all the web pages whose titles contain all the query words
  - Sorts the results by PageRank
  - Very simple and cheap to implement
  - Title match ensures high precision, and PageRank ensures high quality
- *Full Text Search Engine*
  - Used by Google
  - Examines all the words in all the stored document and also performs PageRank(Rank Merging)
  - More precise but more complicated

Implementation

- The damping factor d reflects the probability that the surfer quits the current page and *teleports* to a new one.
- Since every page can be teleported, each page has 1/N probability to be chosen.

PageRank formula based on the mentioned points above

$$PR(p_i) = \frac{1-d}{N} + d \sum_{p_j \in M(p_i)} \frac{PR(p_j)}{L(p_j)}$$

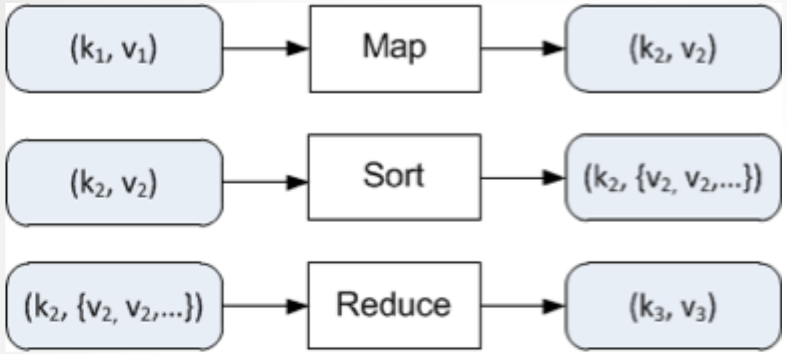
PageRank Algorithm - Pseudocode

```
procedure PageRank(G,iteration)                    #G:inlink file, iteration: # of iteration
   d <- 0.85                                       #damping factor
   oh <- G                                         #get out link count hash from G
   ih <- G                                         #get inlink hash from G
   N <- G                                          #get number of pages from G
   for all p in the graph do
      opg[p] <- 1/N                                #initialise PageRank
   end for
   while iteration > 0 do
      dp <- 0
      for all p in the graph do
         ngp[p] <- dp + (1-d)/N                    #get PageRank from random jump
         for all ip in in ih[p] do
            ngp[p] <- ngp[p] + (d*opg[ip])/oh[ip]  #get PageRank from inlinks
         end for
      end for
      opg <- ngp                                   #update PageRank
      iteration <- iteration - 1
   end while
end procedure
```

MapReduce- What is MapReduce?

- MapReduce is a framework for executing highly parallelizable and distributable algorithms across huge datasets using a large number of commodity computers.
- MapReduce model originates from the map and reduce combinators concept in functional programming languages.
- MapReduce is based on divide and conquer principles.
    - The input datasets are split into independent chunks, which are processed by the mapper in parallel.Execution of the maps is typically co-located with the data.
    - The framework then sorts the outputs of the maps, and uses them as an input to the reducers.

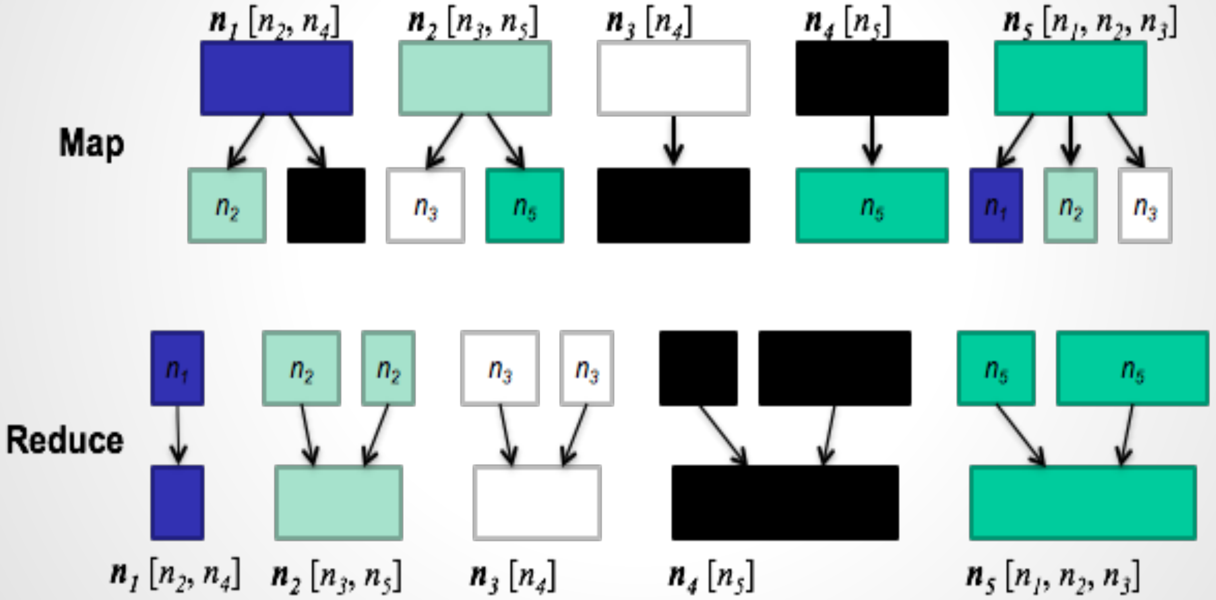
MapReduce in a nutshell



- A mapper takes input in a form of key/value pairs (k1, v1) and transforms them into another key/value pair (k2, v2).
- MapReduce framework sorts mappers output key/value pairs and combines each unique key with all its values (k2, {v2, v2, …}) and deliver these key/value combinations to reducers.
- Reducer takes (k2, {v2, v2, …}) key/value combinations and translates them into yet another key/value pair (k3, v3).

PageRank in MapReduce

Map: distribute PageRank “credit” to link targets



Reduce: gather up PageRank “credit” from multiple sources to compute new PageRank values

Implementing PageRank using MapReduce-Pseudocode

```
map( key : [url, pagerank], value : outlink_list )
   for each outlink in outlink_list :
      emit( key : outlink, value : pagerank / sizeof(outlink_list) )
   emit( key : url, value : outlink_list )

reducer( key : url, value : list_pr_or_urls )
   outlink = [ ]
   pagerank = 0
   for each pr_or_urls in list_pr_or_urls :
      if is_list(pr_or_urls)
         outlink = pr_or_urls
      else
         pagerank += pr_or_urls
   pagerank = 1 - DAMPING_FACTOR + ( DAMPING_FACTOR * pagerank )
   emit( key : [url, pagerank], value : outlink_list )
```

Complexity Analysis of PageRank using MapReduce

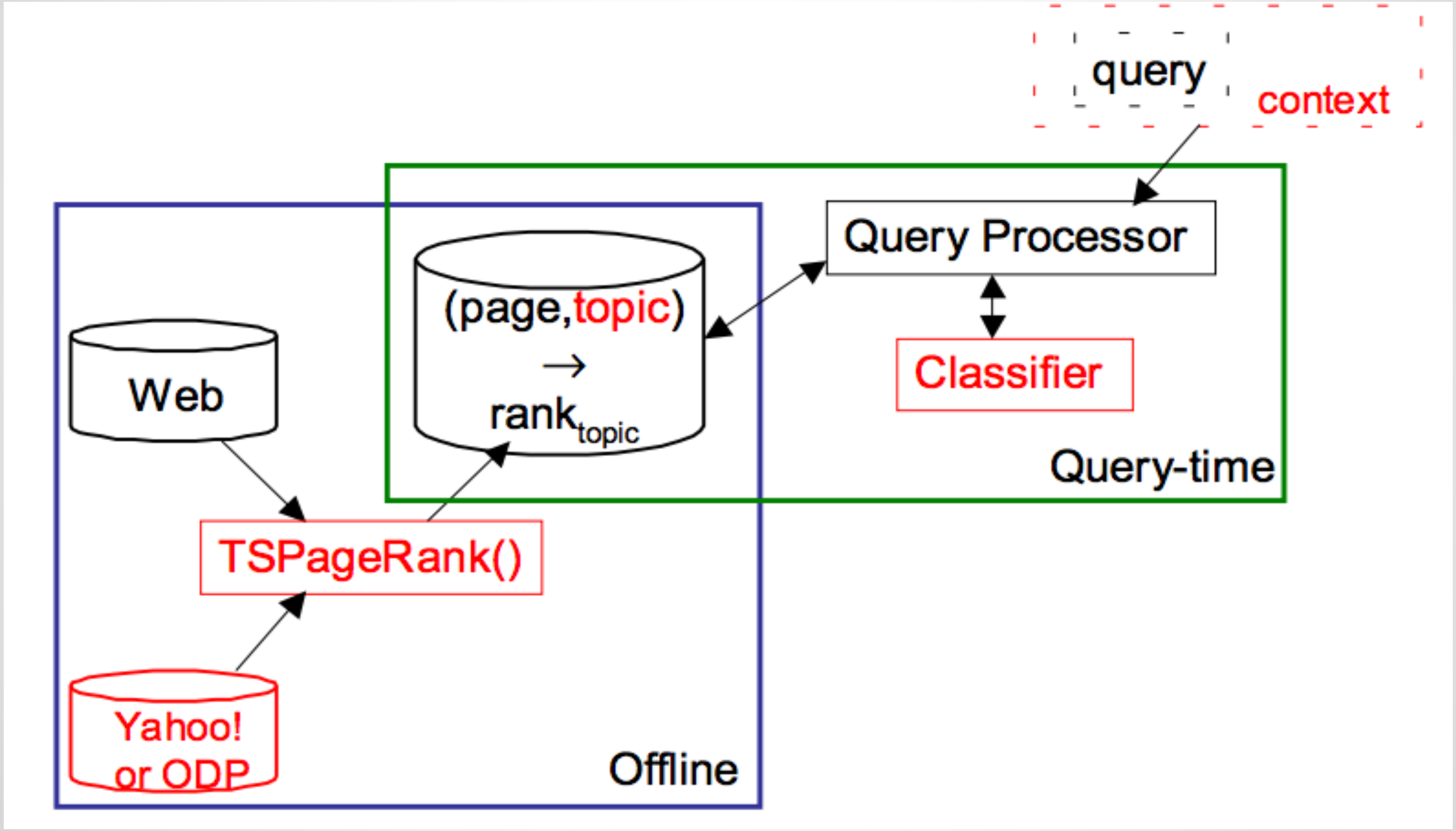
- *Key Complexity*
  - The maximum size of a <KEY, VALUE> pair input to or output by a Mapper/Reducer.
  - The maximum running time for a Mapper/Reducer for a <KEY, VALUE> pair.
  - The maximum memory used by a Mapper/Reducer to process a <KEY, VALUE> pair
- *Sequential Complexity*
  - The size of all <KEY, VALUE> pairs input and output by the Mappers and the Reducers.
  - The total running time for all Mappers and Reducers.

Complexity Analysis of PageRank using MapReduce(2)

Given a directed graph G = (V,E) with M edges, N nodes, and maximum in or out degree d_{max}, each iteration of PageRank (assuming each edge is already annotated with the out-degree of its source node) for batched Reducers is:

- *Key complexity*
  - The size, time, and memory are all $O(d_{MAX})$.
- *Sequential complexity*
  - The total size and running time are both O(M).

Diving Into Topic Sensitive PageRank



Topic Sensitive PageRank

- Different people interested in different topics, expressed using the same query. eg., *Mountain Lion*
- User wants only relevant pages to be given to him for a given query.
- Context sensitive ranking algorithm
- This approach creates one vector for each of some small number of topics, biasing the PageRank to favor pages of that topic.
- Here random surfer surfing for let say “sports” will be directed to a random “sports” page only rather than to a page of any kind.
- *Teleport Set*- Set of pages we have identified to belong to a certain topic
- The mathematical formulation is similar to that of the PageRank with an addition of new surfers

Topic Sensitive PageRank- Formulation

- The mathematical formulation is similar to that of the PageRank with an addition of new surfers
- Let e_s be a vector that has 1 in the components in S and 0 in other components.
- Topic Sensitive PageRank for S is the limit of iteration

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

Here, *M* is the transition matrix of the web, and |*S*| is the size of set *S.*

Integrating TSPR into search engine

1. Decide on the topics for which we shall create specialized PageRank vectors.
2. Pick a teleport set for each of these topics, and use that set to compute the topic-sensitive PageRank vector for that topic.
3. Find a way of determining the topic or set of topics that are most relevant for a particular search query.
4. Use the PageRank vectors for that topic or topics in the ordering of the responses to the search query.

Topic sensitive PageRank Algorithm - Pseudocode

```
procedure PageRank(G,iteration,teleport_set)    #G:inlink file, iteration: # of iteration
    d <- 0.85          #damping factor
    oh <- G            #get out link count hash from G
    ih <- G            #get inlink hash from G
    N <- G             #get number of pages from G
    for all p in the graph do
        opg[p] <- 1        #initialise PageRank
    end for
    while iteration > 0 do
        for all p in the graph do
            if p ϵ teleport_set
                ngp[p] <- (1-d)        #get PageRank from random jump
            for all ip in in ih[p] do
                ngp[p] <- ngp[p] + (d*opg[ip])/oh[ip]    #get PageRank from inlinks
            end for
        end for
        opg <- ngp                                       #update PageRank
        iteration <- iteration - 1
    end while
end procedure
```

TSPR using MapReduce - Pseudocode

```
map( key : [url, pagerank], value : [outlink_list, teleport_list] )
   for each outlink in outlink_list :
      emit( key : outlink, value : pagerank / sizeof(outlink_list) )
   emit( key : url, value : [outlink_list, teleport_list]  )

reducer( key : url, value : list_pr_or_urls )
   outlink = [ ]
   teleport = [ ]
   pagerank = 0
   for each pr_or_urls in list_pr_or_urls :
      if is_list(pr_or_urls)
         outlink = pr_or_urls[0]
        teleport = pr_on_urls[1]
      else
         pagerank += pr_or_urls
   if url ϵ teleport
      pagerank = 1 - DAMPING_FACTOR
   pagerank +=  DAMPING_FACTOR * pagerank
   emit( key : [url, pagerank], value : outlink_list )
```

Complexity Analysis of Topic Sensitive PageRank using MapReduce

- *Key Complexity*
  - The maximum size of a <KEY, VALUE> pair input to or output by a Mapper/Reducer.
  - The maximum running time for a Mapper/Reducer for a <KEY, VALUE> pair.
  - The maximum memory used by a Mapper/Reducer to process a <KEY, VALUE> pair
- *Sequential Complexity*
  - The size of all <KEY, VALUE> pairs input and output by the Mappers and the Reducers.
  - The total running time for all Mappers and Reducers.

Complexity Analysis of Topic Sensitive PageRank using MapReduce(2)

Given a directed graph G = (V,E) with M edges, N nodes, and maximum in or out degree d_{max}, each iteration of PageRank (assuming each edge is already annotated with the out-degree of its source node) for batched Reducers is:

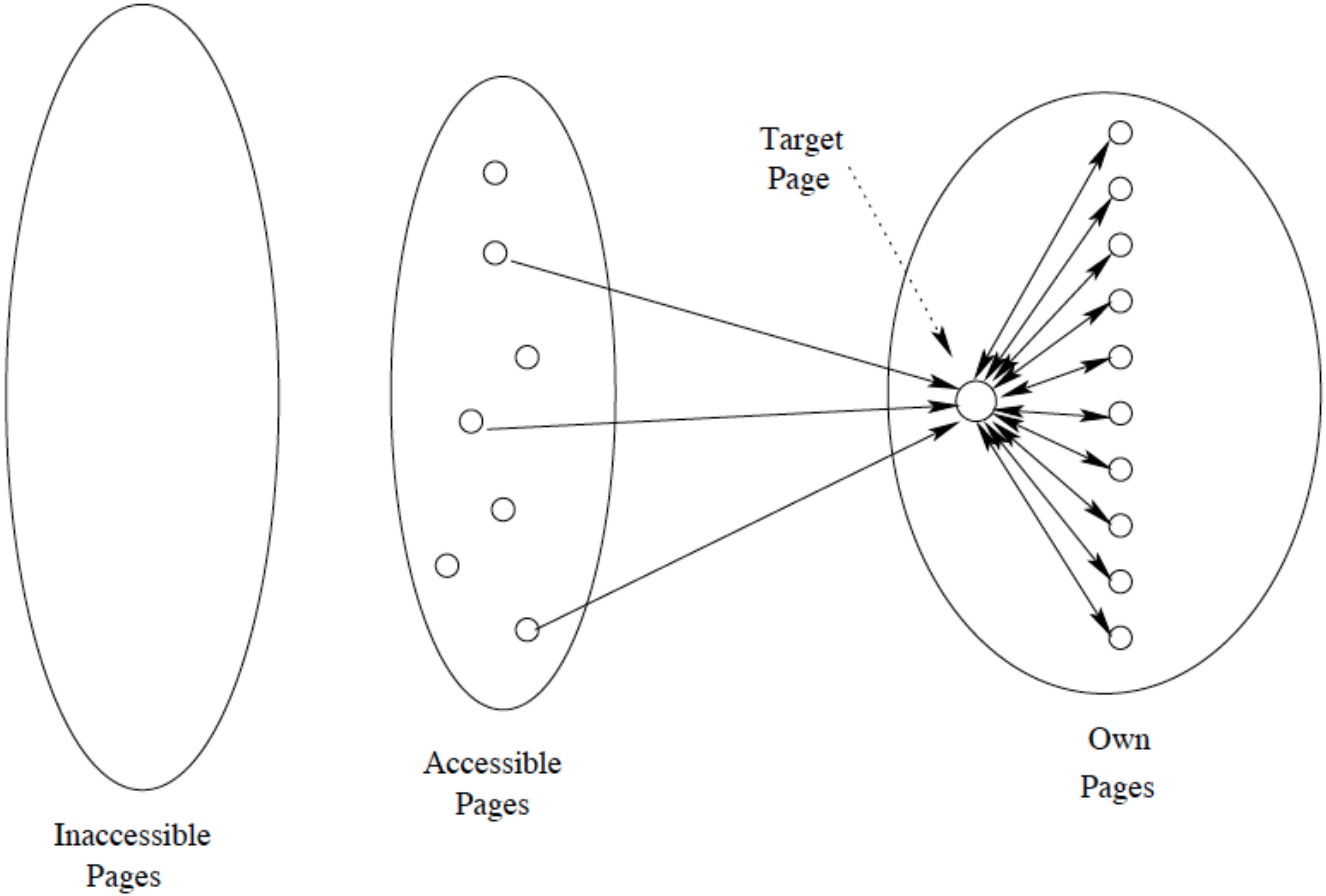
- *Key complexity*
  - The size, time, and memory are all $O(d_{MAX})$.
- *Sequential complexity*
  - The total size and running time are both O(M).

Link Spam

- The techniques for artificially increasing the PageRank of a page are collectively called *link spam*.

Architecture of a Spam Farm

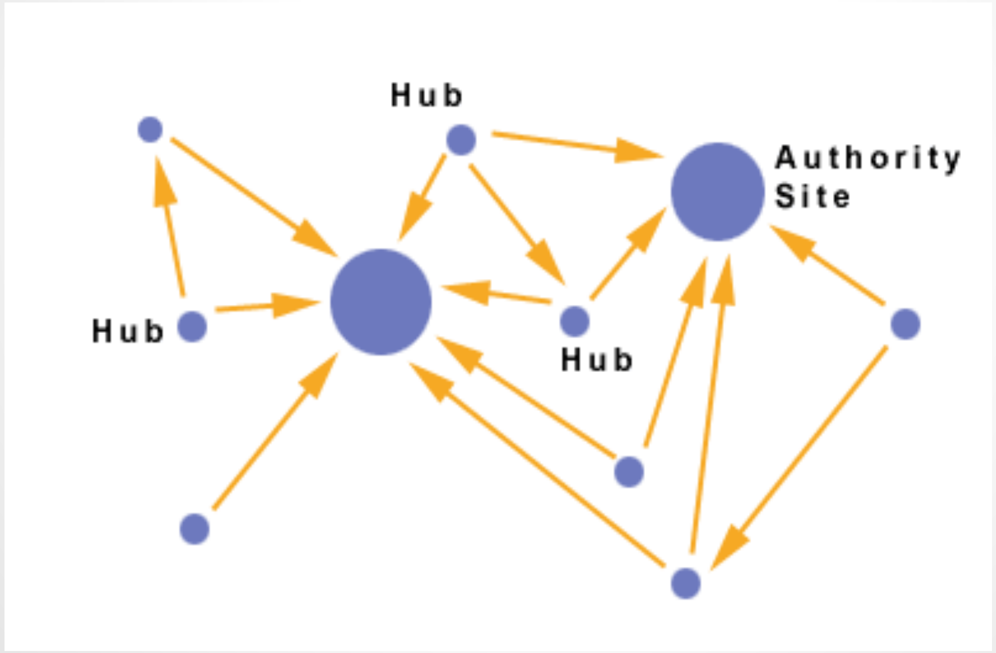
- A collection of pages whose purpose is to increase the PageRank of a certain page or pages is called a *spam farm*.
    - *Inaccessible pages*: the pages that the spammer cannot affect. Most of the Web is in this part.
    - *Accessible pages*: those pages that, while they are not controlled by the spammer, can be affected by the spammer.
    - *Own pages*: the pages that the spammer owns and controls.


Target Page
Inaccessible Pages
Accessible Pages
Own Pages

Combating Link Spam

- One approach is to look for structures like spam farm and eliminate them. This causes spammers to develop different structures to concentrate on their pages.
  - There is essentially no end to variations, so this war between the spammers and the search engines will likely go on for a long time.
- Another approach is to lower the rank of link spam pages automatically via
  - *TrustRank*, a variation of topic-sensitive PageRank designed to lower the score of spam pages.
  - *Spam Mass,* identifies the pages that are likely to be spam and allows the search engine to eliminate those pages or to lower their PageRank strongly.

Diving Into Hubs and Authorities- HITS (Hyperlink-Induced Topic Search)



Hubs and Authorities

Also known as **Hyperlink-Induced Topic search (HITS)** is a scheme in which, given a query, every web page is considered to be of two types:

- Some pages, the most prominent sources of primary content, are the *authorities* on the topic.
- Other pages, assemble high-quality guides and resource lists that act as focused *hubs,* directing users to recommended authorities.
- The nature of the linkage in this framework is highly asymmetric. Hubs link heavily to authorities, but hubs may themselves have very few incoming links, and authorities may well not link to other authorities
- While PageRank assumes a one-dimensional notion of importance of pages, HITS views important pages as having two flavours of importance.

Formalising Hubbiness and Authority

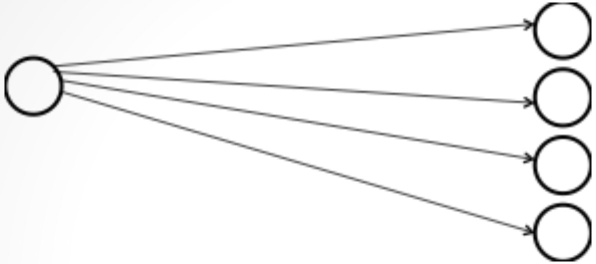
To formalize the intuition, we shall assign two scores to each Web page.

- One score represents the *hubbiness of a page*, that is, the degree to which it is a good hub.
  - Certain pages tell us where to go to find relevant contents about a given topic.
- The second score represents the degree to which the page is a *good authority*.
  - Certain pages are valuable because they provide information about a topic

A page is a good hub if it links to good authorities and a page is a good authority if it is linked to by good hubs.

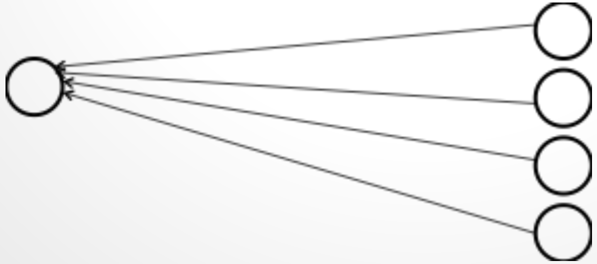
Hub

- A hub is a page with many out-links



Authority

- An authority is a page with many in-links



Example

Suppose we want to buy a car and type in a general query phrase like *"the best automobile makers in the last 4 years"*.

Official web sites of car manufacturers, such as **www.bmw.com**, **www. HyundaiUSA.com**, **www.mercedes-benz.com**, would be authorities for this search.

However, there is a second category of pages relevant to the process of finding the authoritative pages, called ***hubs***. Their role is to advertise the authoritative pages.

PageRank vs Hits

PageRank	Hits
Computed for all web pages stored prior to the query	Performed on the subset generated by each query
Computes authorities only	Computes both authorities and hubs
Fast to compute	Easy to compute, real time execution is hard

Hubbiness and Authority

- To estimate Hubbiness add authorities of successors.

$h = \lambda L a$

where L= Link Matrix

λ= constant

a= authority

- To estimate Authorities add hubbiness of predecessors.

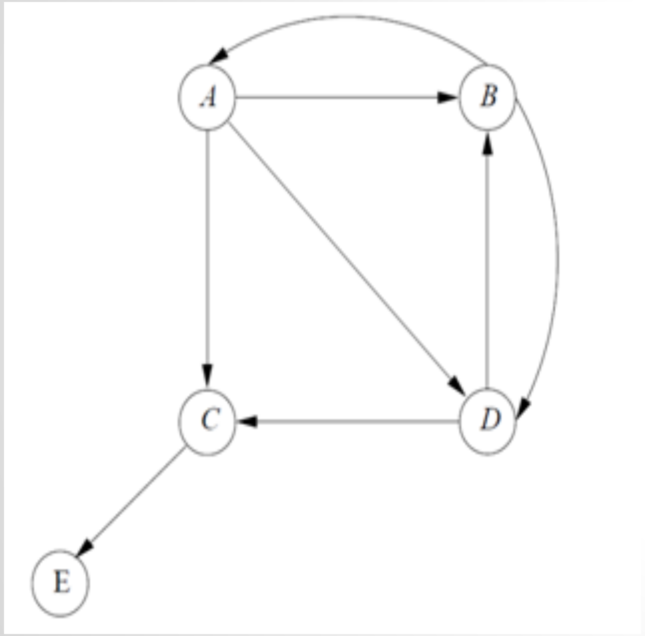
$a = \mu L^T h$

where L^T = Transpose of Link Matrix

h = hubs

μ= constant

Example



$$L = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$L^{\mathrm{T}} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

Example (contd.)

First two iterations of the HITS Algorithm

$$\begin{array}{ccccc}
\begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} & \begin{bmatrix} 1 \\ 2 \\ 2 \\ 2 \\ 1 \end{bmatrix} & \begin{bmatrix} 1/2 \\ 1 \\ 1 \\ 1 \\ 1/2 \end{bmatrix} & \begin{bmatrix} 3 \\ 3/2 \\ 1/2 \\ 2 \\ 0 \end{bmatrix} & \begin{bmatrix} 1 \\ 1/2 \\ 1/6 \\ 2/3 \\ 0 \end{bmatrix} \\
\mathbf{h} & L^{\mathrm{T}}\mathbf{h} & \mathbf{a} & L\mathbf{a} & \mathbf{h}
\end{array}$$

$$\begin{array}{cccc}
\begin{bmatrix} 1/2 \\ 5/3 \\ 5/3 \\ 3/2 \\ 1/6 \end{bmatrix} & \begin{bmatrix} 3/10 \\ 1 \\ 1 \\ 9/10 \\ 1/10 \end{bmatrix} & \begin{bmatrix} 29/10 \\ 6/5 \\ 1/10 \\ 2 \\ 0 \end{bmatrix} & \begin{bmatrix} 1 \\ 12/29 \\ 1/29 \\ 20/29 \\ 0 \end{bmatrix} \\
L^{\mathrm{T}}\mathbf{h} & \mathbf{a} & L\mathbf{a} & \mathbf{h}
\end{array}$$

HITS - Pseudocode

```
G := set of pages
for each page p in G do
   p.auth = 1    // p.auth is the authority score of the page p
   p.hub = 1    // p.hub is the hub score of the page p
function HubsAndAuthorities(G)
   for step from 1 to k do    // run the algorithm for k steps
      norm = 0
      for each page p in G do     // update all authority values first
         p.auth = 0
         for each page q in p.incomingNeighbors do    // p.incomingNeighbors is the set of pages that link to p
            p.auth += q.hub
            norm += square(p.auth)    // calculate the sum of the squared auth values to normalise
      norm = sqrt(norm)
      for each page p in G do     // update the auth scores
         p.auth = p.auth / norm     // normalise the auth values
      norm = 0
      for each page p in G do     // then update all hub values
         p.hub = 0
         for each page r in p.outgoingNeighbors do    // p.outgoingNeighbors is the set of pages that p links to
            p.hub += r.auth
            norm += square(p.hub)    // calculate the sum of the squared hub values to normalise
      norm = sqrt(norm)
      for each page p in G do     // then update all hub values
         p.hub = p.hub / norm      // normalise the hub values
```

HITS Using MapReduce- Pseudocode

MAP

```
map( key: [url, auth, hub], value: [inlink_list, outlink_list] )
    for each outlink in outlink_list :
        emit( key : outlink, value : [hub, “auth”] )
      //page taking its auth value from predecessor hub
    for each inlink in inlink_list :
        emit( key : inlink, value : [auth, “hub”] )
      //page taking its hub value from successor auth
    emit( key : url, value : [inlink_list, outlink_list] )
```

HITS Using MapReduce Pseudocode

REDUCE

```
reducer( key : url, value : list_link_auth_hub )
   outlink_list = [ ]
   inlink_list = [ ]
   auth = 0
   hub = 0
   for each link_auth_hub in list_link_auth_hub :
      if is_list( link_auth_hub[1] )
         inlink_list = link_auth_hub[0]
         outlink_list = link_auth_hub[1]
   else
      if ( link_auth_hub[1] == “auth” )
         auth += link_auth_hub[0]
      else
         hub += link_auth_hub[0]
   emit(key : [url, auth, hub], value : [inlink_list, outlink_list])
```

Complexity Analysis of HITS using MapReduce

- *Key Complexity*
  - The maximum size of a <KEY, VALUE> pair input to or output by a Mapper/Reducer.
  - The maximum running time for a Mapper/Reducer for a <KEY, VALUE> pair.
  - The maximum memory used by a Mapper/Reducer to process a <KEY, VALUE> pair
- *Sequential Complexity*
  - The size of all <KEY, VALUE> pairs input and output by the Mappers and the Reducers.
  - The total running time for all Mappers and Reducers.

Complexity Analysis of HITS using MapReduce(2)

Given a directed graph G = (V,E) with M edges, N nodes, and maximum in or out degree d_{max}, each iteration of PageRank (assuming each edge is already annotated with the out-degree of its source node) for batched Reducers is:

- *Key complexity*
  - The size, time, and memory are all $O(d_{MAX})$.
- *Sequential complexity*
  - The total size and running time are both O(M).