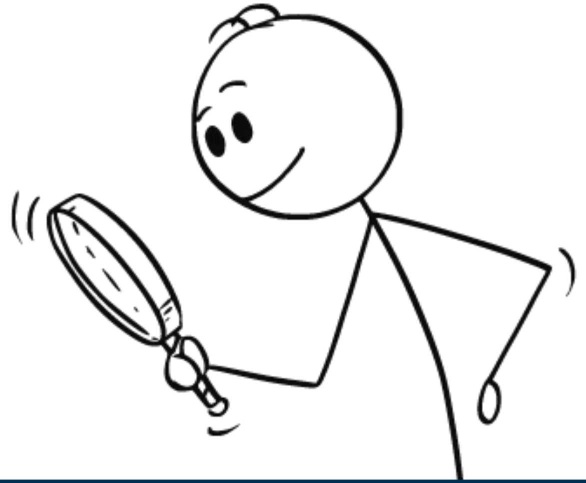


CSE 4/535
Information Retrieval

Sayantan Pal
PhD Student, Department of CSE
338Z Davis Hall

University at Buffalo
Department of CSE

Before we start

1. Project 2 is due today - 11:59 PM
2. Make sure to keep your VMs running unless instructed to shut down
3. Today's lecture
   a. Connectivity Servers
   b. Link Analysis (Important for Midterm 2)
      i. Anchor text
      ii. Page Rank

Recap - Previous Class

1. Web Search and Crawling
    a. Shingles and Sketches
    b. Mercator Scheme
    c. Front Queues, Back Queues

Connectivity servers

Connectivity Server

[*CS1:* Bhar98b, *CS2 & 3:* Rand01]

- Support for fast queries on the web graph
  - Which URLs point to a given URL?
  - Which URLs does a given URL point to?

Stores mappings in memory from

- URL to outlinks, URL to inlinks

- Applications
  - Crawl control
  - Web graph analysis
    - Connectivity, crawl optimization
  - Link analysis

20.4

Champion published work

- Boldi and Vigna
  - http://www2004.org/proceedings/docs/1p595.pdf
- Webgraph – set of algorithms and a java implementation
- Fundamental goal – maintain node adjacency lists in memory
  - For this, compressing the adjacency lists is the critical component

Adjacency lists

- The set of neighbors of a node
- Assume each URL represented by an integer
- E.g., for a 4 billion page web, need 32 bits per node
- Naively, this demands 64 bits to represent each hyperlink

Adjaceny list compression

- Properties exploited in compression:
  - Similarity (between lists)
  - Locality (many links from a page go to “nearby” pages)
  - Use gap encodings in sorted lists
  - Distribution of gap values

Storage

- Boldi/Vigna get down to an average of ~3 bits/link
  - (URL to URL edge)
  - For a 118M node web graph
- How?

Why is this remarkable?

Main ideas of Boldi/Vigna

- Consider lexicographically ordered list of all URLs, e.g.,
  - www.stanford.edu/alchemy
  - www.stanford.edu/biology
  - www.stanford.edu/biology/plant
  - www.stanford.edu/biology/plant/copyright
  - www.stanford.edu/biology/plant/people
  - www.stanford.edu/chemistry

Boldi/Vigna: Adjacency Table

- Each of these URLs has an adjacency list
- Main idea: due to templates, the adjacency list of a node is similar to one of the 7 preceding URLs in the lexicographic ordering
- Express adjacency list in terms of one of these
- E.g., consider these adjacency lists
  - 1, 2, 4, 8, 16, 32, 64
  - 1, 4, 9, 16, 25, 36, 49, 64
  - 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144
  - 1, 4, 8, 16, 25, 36, 49, 64

Why 7?

Each row represents a URL

Encode as (-2), remove 9, add 8

What if none of the previous rows are a good prototype?

Boldi/Vigna: Adjacency Table

- Each of these URLs has an adjacency list
- Main idea: due to templates, the adjacency list of a node is similar to one of the 7 preceding URLs in the lexicographic ordering

Why 7?

- Express adjacency list in terms of one of these
- E.g., consider these adjacency lists
  - 1, 2, 4, 8, 16, 32, 64
  - 1, 4, 9, 16, 25, 36, 49, 64
  - 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144
  - 1, 4, 8, 16, 25, 36, 49, 64

Each row represents a URL

Encode as (-2), remove 9, add 8

What if none of the previous rows are a good prototype?

3 bit representation, gap encoding

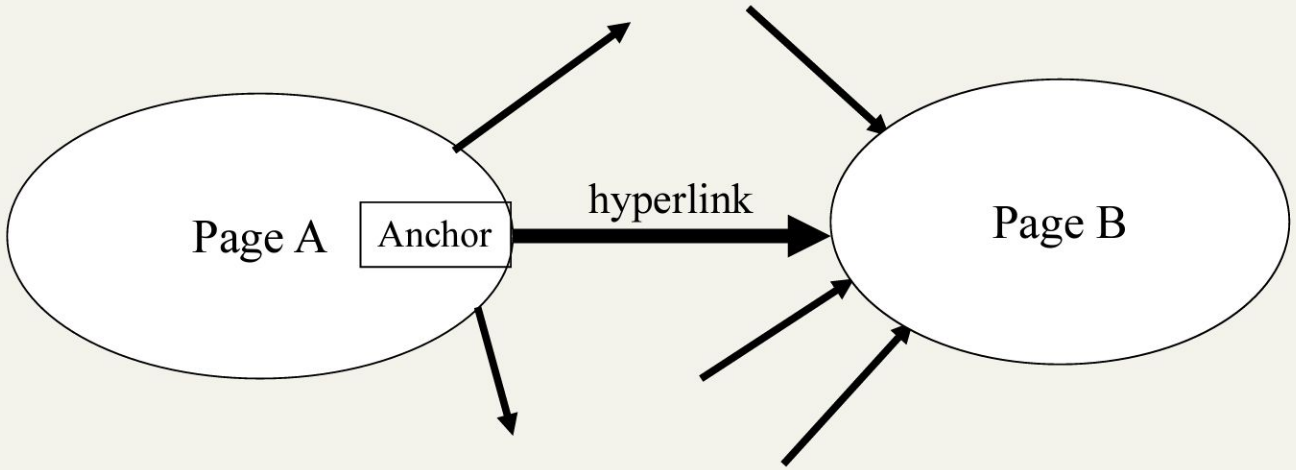
Why 7? 3 bit representation, first 7 numbers used to specify which of the previous 7 it is similar to; need to reserve integer 8 in case it is not similar to any preceding rows.

Encode as 3-bit offset, integers representing which nodes to be removed, and added

If no rows are similar, start a new list, using gap encoding between pages

Link Analysis

The Web as a Directed Graph

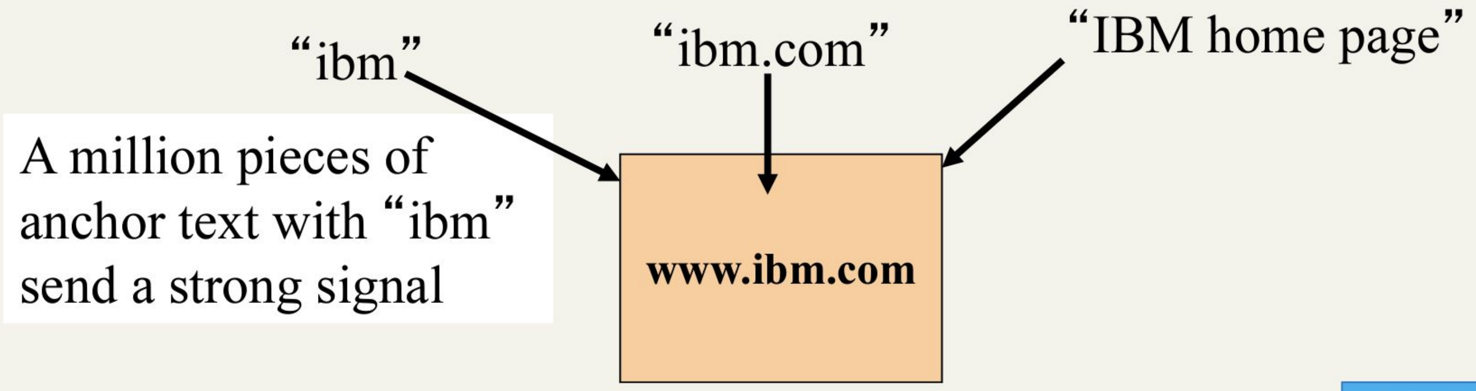


Assumption 1: A hyperlink between pages denotes author perceived relevance (quality signal)

Assumption 2: The anchor of the hyperlink describes the target page (textual context)

Anchor Text

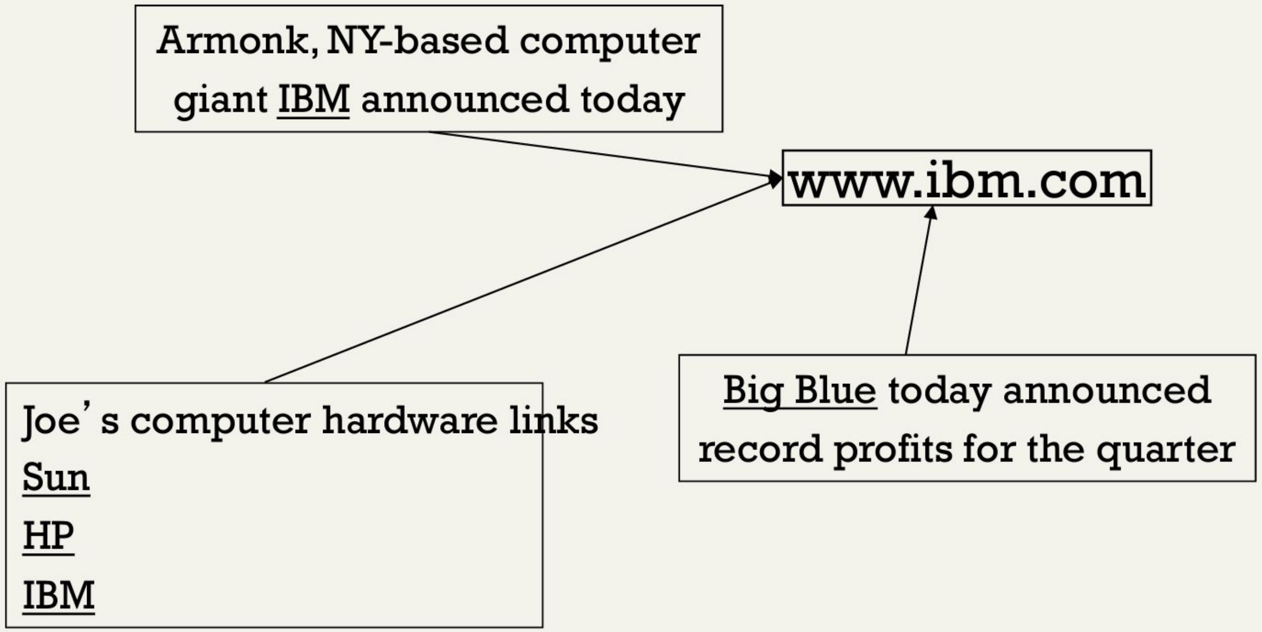
- For ***ibm*** how to distinguish between:
  - IBM's home page (mostly graphical)
  - IBM's copyright page (high term freq. for 'ibm')
  - Rival's spam page (arbitrarily high term freq.)



21.1.1

Indexing anchor text

- When indexing a document *D*, include anchor text from links pointing to *D*.



Indexing anchor text

- Can sometimes have unexpected side effects - *e.g.,* ***evil empire***.
- Can score anchor text with weight depending on the authority of the anchor page's website
  - E.g., if we were to assume that content from cnn.com or yahoo.com is authoritative, then trust the anchor text from them

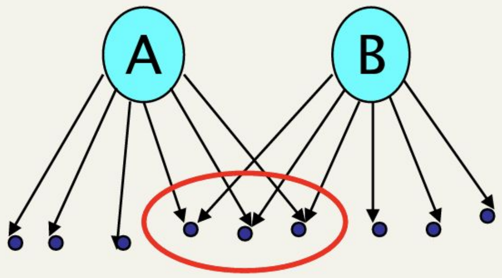
Anchor Text

- Other applications
  - Weighting/filtering links in the graph
  - Generating page descriptions from anchor text

University at Buffalo
Department of CSE

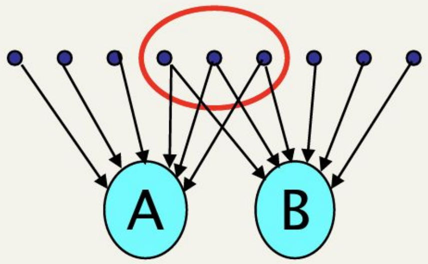
Bibliographic Coupling

- Measure of similarity of documents introduced by Kessler in 1963.
- The bibliographic coupling of two documents *A* and *B* is the number of documents cited by *both A* and *B*.
- Size of the intersection of their bibliographies.
- Maybe want to normalize by size of bibliographies?



Co-Citation

- An alternate citation-based measure of similarity introduced by Small in 1973.
- Number of documents that cite both *A* and *B*.
- Maybe want to normalize by total number of documents citing either *A* or *B* ?

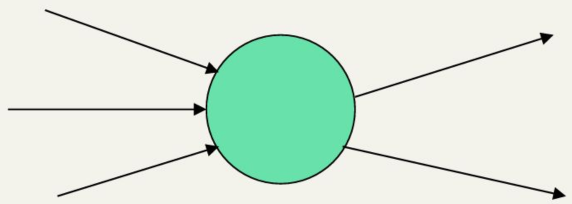


Citations vs. Links

- Web links are a bit different than citations:
  - Many links are navigational.
  - Many pages with high in-degree are portals not content providers.
  - Not all links are endorsements.
  - Company websites don't point to their competitors.
  - Citations to relevant literature is enforced by peer-review.

Query-independent ordering

- First generation: using link counts as simple measures of popularity.
- Two basic suggestions:
  - Undirected popularity:
    - Each page gets a score = the number of in-links plus the number of out-links (3+2=5).
  - Directed popularity:
    - Score of a page = number of its in-links (3).



Query processing

- First retrieve all pages meeting the text query (say ***venture capital***).
- Order these by their link popularity (either variant on the previous page).

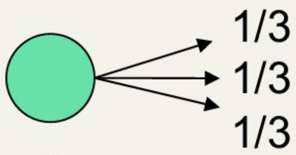
- *** can use Boolean model for step 1 ***

Spamming simple popularity

- *Exercise*: How do you spam each of the following heuristics so your page gets a high score?
- Each page gets a static score = the number of in-links plus the number of out-links.
- Static score of a page = number of its in-links.

Pagerank scoring

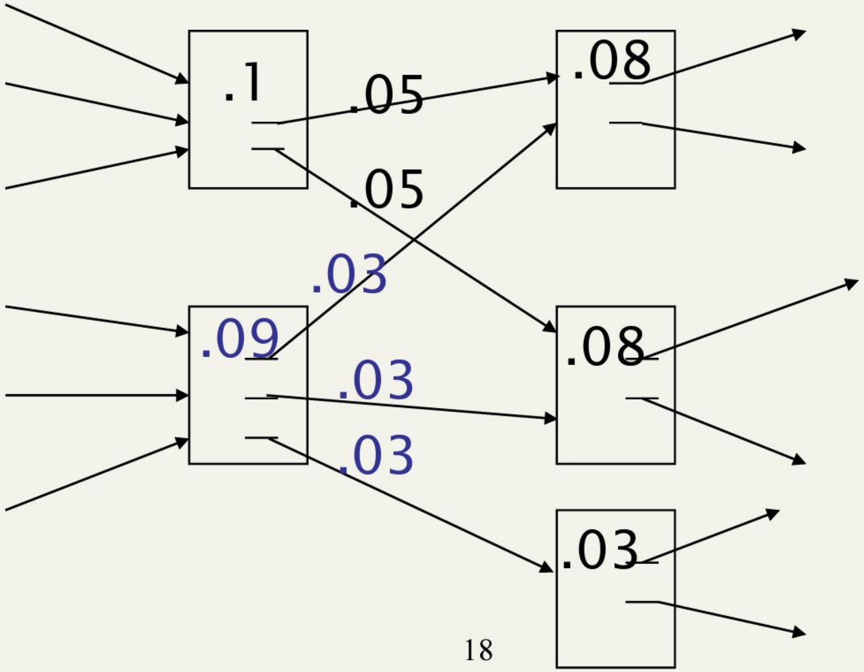
- Imagine a browser doing a random walk on web pages:
  - Start at a random page



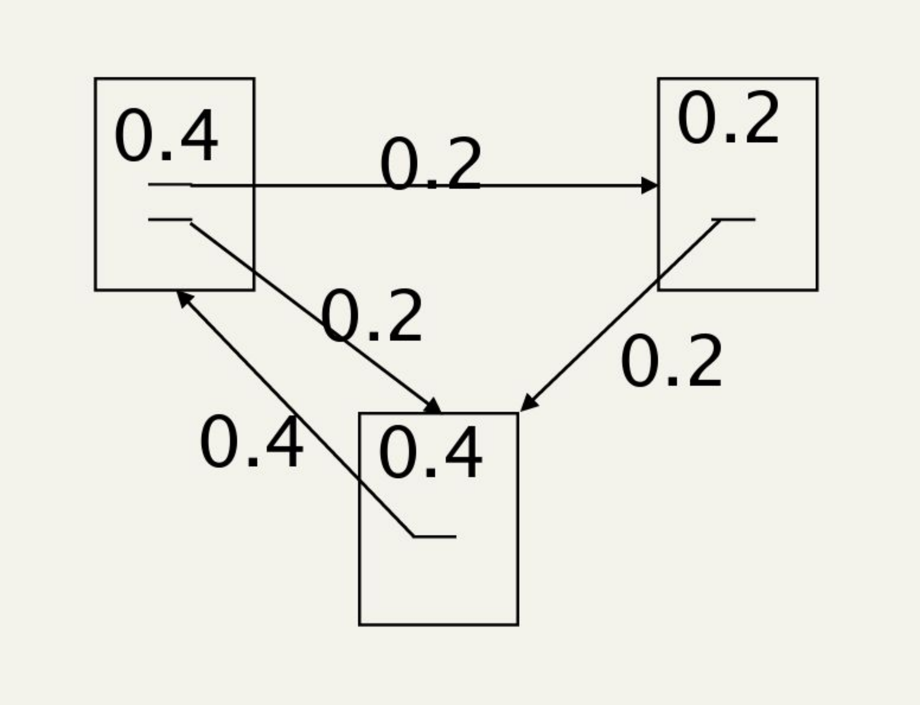
  - At each step, go out of the current page along one of the links on that page, equiprobably
- "In the steady state" each page has a long-term visit rate - use this as the page's score.

Initial PageRank Idea

- Can view it as a process of PageRank "flowing" from pages to the pages they cite.

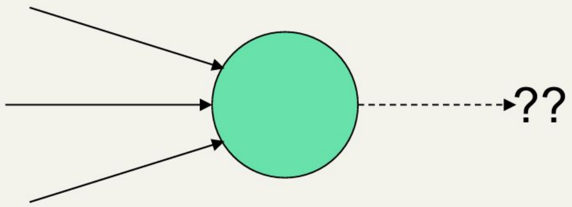


Sample Stable Fixpoint



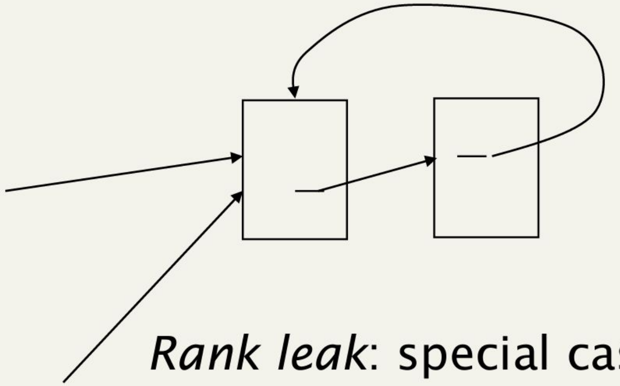
Not quite enough

- The web is full of dead-ends.
  - Random walk can get stuck in dead-ends.
  - Makes no sense to talk about long-term visit rates.



Problem with Initial Idea

- A group of pages that only point to themselves but are pointed to by other pages act as a “rank sink” and absorb all the rank in the system.



Rank flows into
cycle and can’t get out

Rank leak: special case of rank sink where

an individual page does not have any outlinks

Teleporting

- At a dead end, jump to a random web page.
- At any non-dead end, with probability 10%, jump to a random web page.
  - With remaining probability (90%), go out on a random link.
  - 10% - a parameter.

Result of teleporting

- Now cannot get stuck locally.
- There is a long-term rate at which any page is visited (not obvious, will show this).
- How do we compute this visit rate?

Markov chains

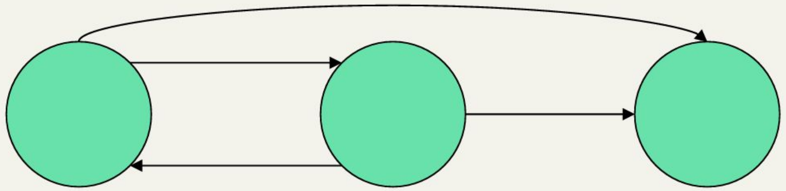
- A Markov chain consists of n states, plus an $n \times n$ transition probability matrix **P**.
- At each step, we are in exactly one of the states.
- For $1 \leq i,j \leq n$, the matrix entry P_{ij} tells us the probability of j being the next state, given we are currently in state i.

$P_{ii}>0$ is OK.

Markov chains

- Clearly, for all i, $\sum_{j=1}^{n} P_{ij} = 1.$
- Markov chains are abstractions of random walks.
- *Exercise*: represent the teleporting random walk from 3 slides ago as a Markov chain, for this case:

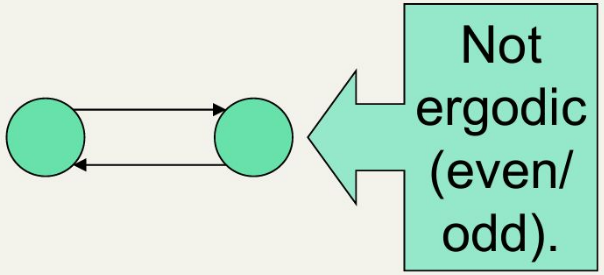
Hint: Matrix of transition probabilities



21.2.1

Ergodic Markov chains

- A Markov chain is ergodic if
  - you have a path from any state to any other
  - For any start state, after a finite transient time T_0, the probability of being in any state at a fixed time $T>T_0$ is nonzero.



21.2.1

- Recurrent
- Aperiodic

Ergodic Markov chains

- Theorem: For any ergodic Markov chain, there is a unique long-term visit rate for each state.
  - *Steady-state probability distribution*.
- Over a long time-period, we visit each state in proportion to this rate.
- It doesn't matter where we start.

Probability vectors

- A probability (row) vector $\mathbf{x} = (x_1, \ldots x_n)$ tells us where the walk is at any point.
- E.g., $(\underset{1}{0}00\ldots\underset{i}{1}\ldots00\underset{n}{0})$ means we're in state i.

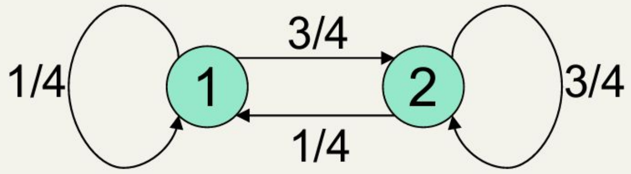
More generally, the vector $\mathbf{x} = (x_1, \ldots x_n)$ means the walk is in state i with probability x_i.

$$\sum_{i=1}^{n} x_i = 1.$$

21.2.1

Steady state example

- The steady state looks like a vector of probabilities **a** = $(a_1, \dots a_n)$:
  - a_i is the probability that we are in state *i*.



For this example, $a_1=1/4$ and $a_2=3/4$.

PageRank Calculation

PR(A)=(1-d) + d*(PR(T1)/C(T1)+…+ PR(Tn)/C(Tn))

d: damping factor, normally this is set to 0.85.

T1, …, Tn: pages pointing to page A

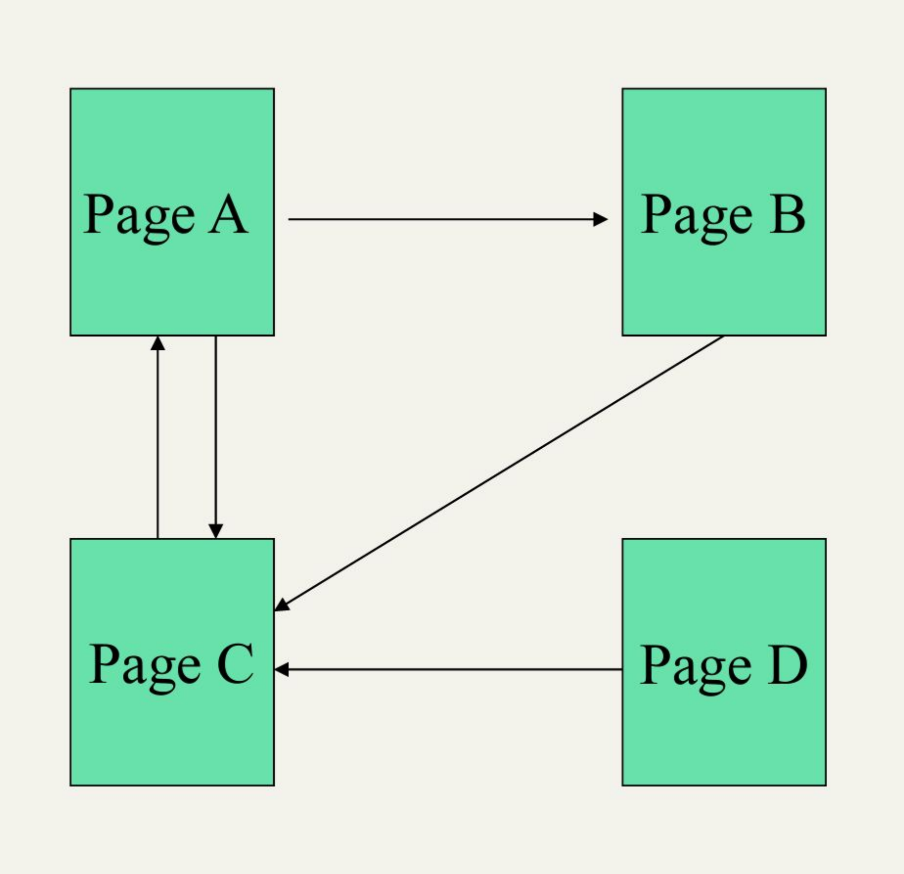
PR(A): PageRank of page A.

PR(Ti): PageRank of page Ti.

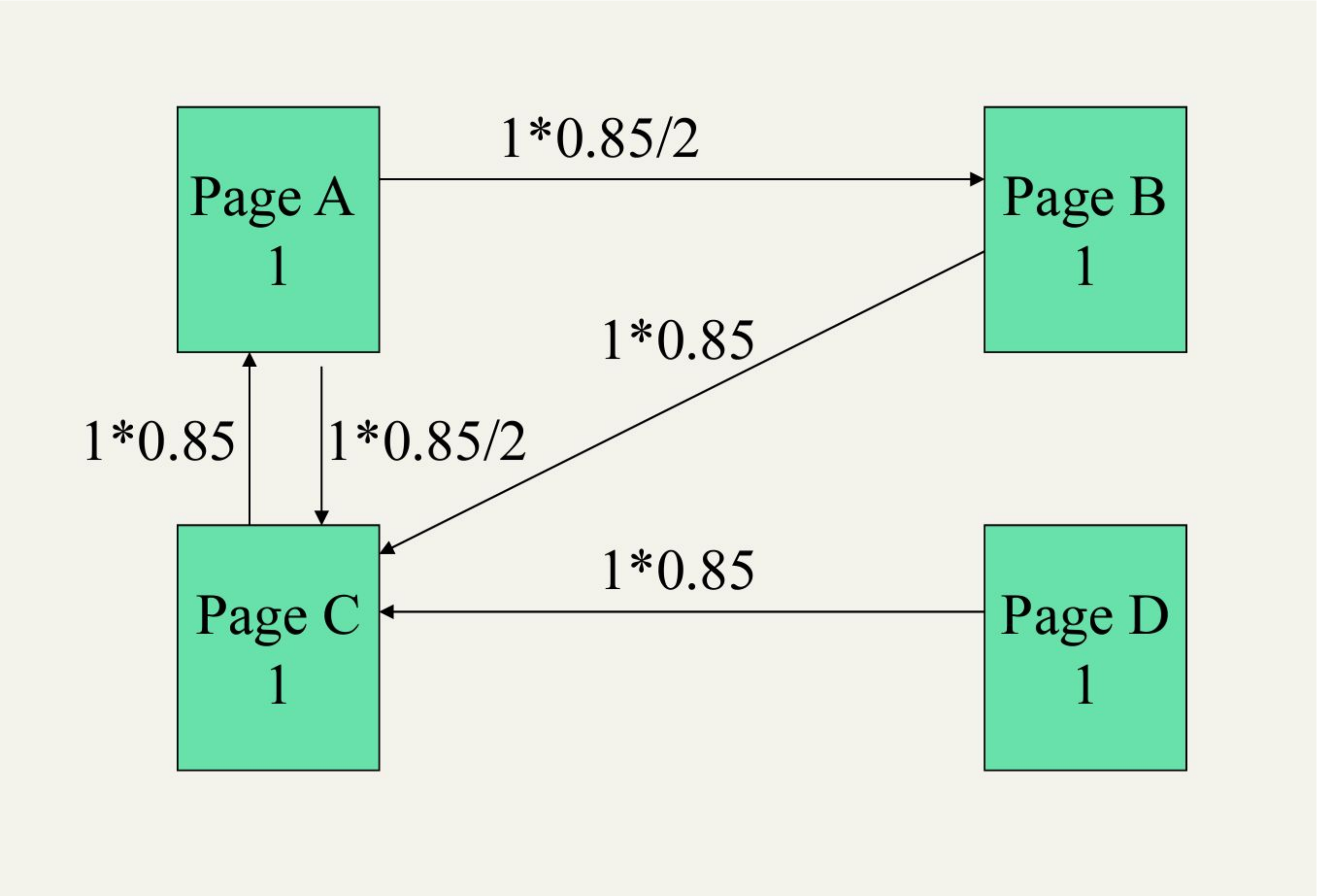
C(Ti): the number of links going out of page Ti.

Note: d counts for PageRank sinks

Example of Calculation

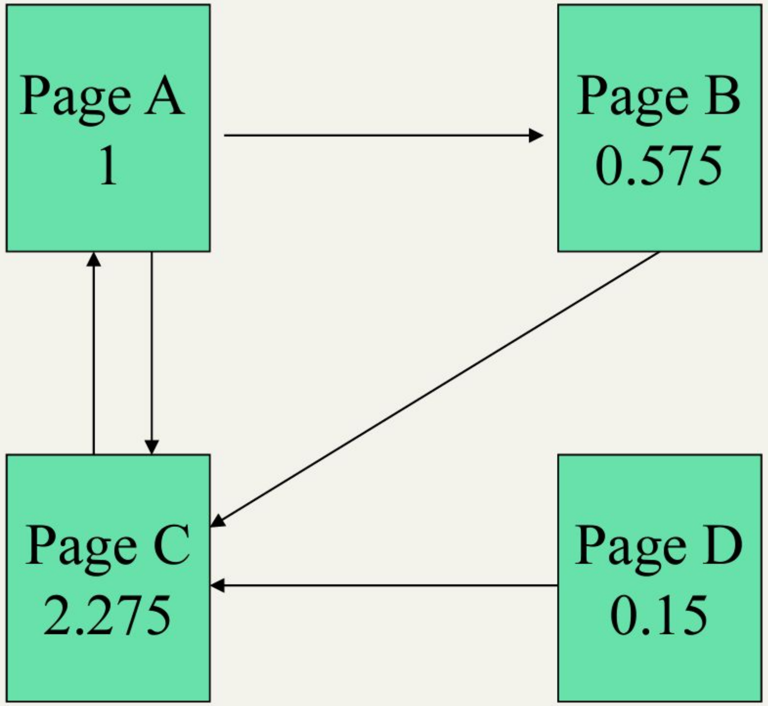


Example of Calculation



- Assume d = 0.85

Example of Calculation



- Each page has not passed on 0.15, so we get:

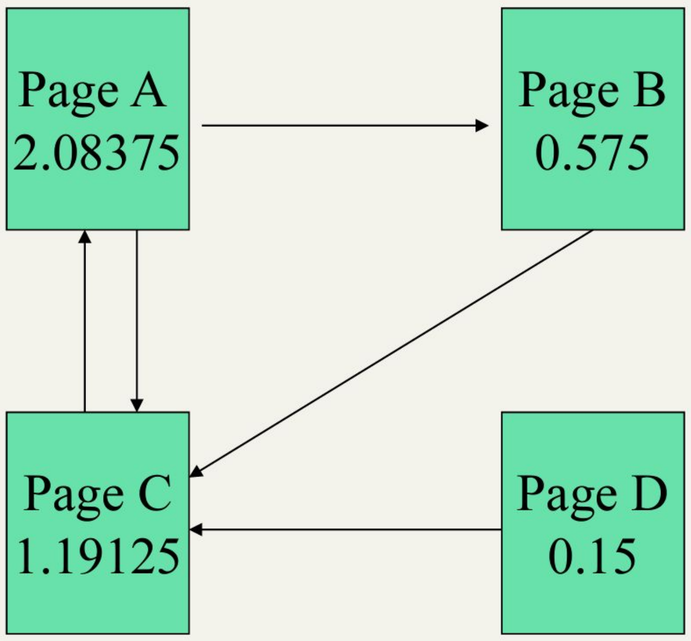
  Page A: 0.85 (from Page C) + 0.15 (not transferred) = 1

  Page B: 0.425 (from Page A) + 0.15 (not transferred) = 0.575

  Page C: 0.85 (from Page D) + 0.85 (from Page B) + 0.425 (from Page A) + 0.15 (not transferred) = 2.275

  Page D: receives none, but has not transferred 0.15 = 0.15

Example of Calculation



Page A: 2.275*0.85 (from Page C) + 0.15 (not transferred) =

2.08375

Page B: 1*0.85/2 (from Page A) + 0.15 (not transferred) =

0.575

Page C: 0.15*0.85 (from Page D) + 0.575*0.85(from Page B) + 1*0.85/2 (from Page A) +0.15 (not transferred) =

1.19125

Page D: receives none, but has not transferred 0.15 = 0.15

Example -Conclusions

- Page C has the highest PageRank, and page A has the next highest: page C has a highest importance in this page graph!
- More iterations lead to convergence of PageRanks.

How do we compute this vector?

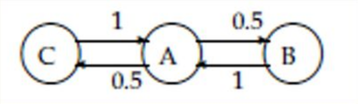
- Let **a** = $(a_1, \ldots a_n)$ denote the row vector of steady-state probabilities.
- If we our current position is described by **a**, then the next step is distributed as **aP**.
- But **a** is the steady state, so **a**=**aP**.
- Solving this matrix equation gives us **a**.
  - So **a** is the (left) eigenvector for **P**.
  - (Corresponds to the "principal" eigenvector of **P** with the largest eigenvalue.)
  - Transition probability matrices always have largest eigenvalue 1.

21.2.2

One way of computing 'a'

- Recall, regardless of where we start, we eventually reach the steady state **a**.
- Start with any distribution (say **x**=(*10…0*)).
- After one step, we're at **xP**;
- after two steps at $\mathbf{xP}^2$, then $\mathbf{xP}^3$ and so on.
- "Eventually" means for "large" k, $\mathbf{xP}^k = \mathbf{a}$.
- Algorithm: multiply **x** by increasing powers of **P** until the product looks stable.

Transition Prob Matrix



$$\begin{pmatrix} 0 & 0.5 & 0.5 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix}$$

Transition Probability Matrix

Deriving Transition Probability Matrix from Adjacency Matrix *A*

1. If a row of A has no 1's, then replace each element by $1/N$. For all other rows proceed as follows.
2. Divide each 1 in A by the number of 1's in its row. Thus, if there is a row with three 1's, then each of them is replaced by $1/3$.
3. Multiply the resulting matrix by $1-\alpha$.
4. Add α/N to every entry of the resulting matrix, to obtain P.

$$P = \begin{pmatrix} 1/6 & 2/3 & 1/6 \\ 5/12 & 1/6 & 5/12 \\ 1/6 & 2/3 & 1/6 \end{pmatrix}.$$

Transition Prob Matrix for alpha = 0.5

Power Iterations

$$P = \begin{pmatrix} 1/6 & 2/3 & 1/6 \\ 5/12 & 1/6 & 5/12 \\ 1/6 & 2/3 & 1/6 \end{pmatrix}.$$

Imagine that the surfer starts in state 1, corresponding to the initial probability distribution vector $\vec{x_0} = (1\ 0\ 0)$. Then, after one step the distribution is

$$\vec{x_0}P = \begin{pmatrix} 1/6 & 2/3 & 1/6 \end{pmatrix} = \vec{x_1}.$$

After two steps it is

$$\vec{x_1}P = \begin{pmatrix} 1/6 & 2/3 & 1/6 \end{pmatrix} \begin{pmatrix} 1/6 & 2/3 & 1/6 \\ 5/12 & 1/6 & 5/12 \\ 1/6 & 2/3 & 1/6 \end{pmatrix} = \begin{pmatrix} 1/3 & 1/3 & 1/3 \end{pmatrix} = \vec{x_2}.$$

$\vec{x_0}$	1	0	0
$\vec{x_1}$	1/6	2/3	1/6
$\vec{x_2}$	1/3	1/3	1/3
$\vec{x_3}$	1/4	1/2	1/4
$\vec{x_4}$	7/24	5/12	7/24
...	...	...	...
$\vec{x}$	5/18	4/9	5/18

Steady State

Pagerank summary

- Preprocessing:
  - Given graph of links, build matrix **P**.
  - From it compute **a**.
  - The entry a_i is a number between 0 and 1: the pagerank of page *i*.
- Query processing:
  - Retrieve pages meeting query.
  - Rank them by their pagerank.
  - Order is query-*independent*.

Other applications of PageRank

- LexRank
  - Sentence ranking for multidocument summarization
    http://www.cs.cmu.edu/afs/cs/project/jair/pub/volume22/erkan04a-html/erkan04a.html
- TextRank (Mihalcea)
  - Graph-based ranking model for graphs extracted from natural language texts : word extraction, sentence extraction. More general.

References

1. Slides provided by Sougata Saha (Instructor, Fall 2022 - CSE 4/535)
2. Materials provided by Dr. Rohini K Srihari
3. https://nlp.stanford.edu/IR-book/information-retrieval-book.html