Ranking web pages

- Web pages are not equally "important"
  - www.joe-schmoe.com v www.stanford.edu
- Inlinks as votes
  - www.stanford.edu has 23,400 inlinks
  - www.joe-schmoe.com has 1 inlink
- Are all inlinks equal?
  - Recursive question!

Simple recursive formulation

- Each link’s vote is proportional to the importance of its source page
- If page P with importance x has n outlinks, each link gets x/n votes

Simple “flow” model

The web in 1839

\[ y = y/2 + a/2 \]
\[ a = y/2 + m \]
\[ m = a/2 \]

Solving the flow equations

- 3 equations, 3 unknowns, no constants
  - No unique solution
  - All solutions equivalent modulo scale factor
- Additional constraint forces uniqueness
  - \( y + a + m = 1 \)
  - \( y = 2/5, a = 2/5, m = 1/5 \)
- Gaussian elimination method works for small examples, but we need a better method for large graphs
Matrix formulation
- Matrix $M$ has one row and one column for each web page.
- Suppose page $j$ has $n$ outlinks.
  - If $j \rightarrow i$, then $M_{ij} = 1/n$.
  - Else $M_{ij} = 0$.
- $M$ is a column stochastic matrix.
  - Columns sum to 1.
- Suppose $r$ is a vector with one entry per web page.
  - $r_i$ is the importance score of page $i$.
  - Call it the rank vector.

Eigenvector formulation
- The flow equations can be written $r = Mr$.
- So the rank vector is an eigenvector of the stochastic web matrix.
  - In fact, its first or principal eigenvector, with corresponding eigenvalue 1.

Power Iteration method
- Simple iterative scheme (aka relaxation).
- Suppose there are $N$ web pages.
- Initialize: $r^0 = [1/N, ..., 1/N]^T$.
- Iterate: $r^{k+1} = Mr^k$.
- Stop when $|r^{k+1} - r^k|_1 < \varepsilon$.
  - $|x|_1 = \sum |x_i|$ is the $L_1$ norm.
  - Can use any other vector norm e.g., Euclidean.

Example
- Suppose page $j$ links to 3 pages, including $i$.

### Example

<table>
<thead>
<tr>
<th></th>
<th>y</th>
<th>a</th>
<th>m</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>1/2</td>
<td>1/2</td>
<td>0</td>
</tr>
<tr>
<td>a</td>
<td>1/2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>m</td>
<td>0</td>
<td>1/2</td>
<td>0</td>
</tr>
</tbody>
</table>

$y = y/2 + a/2$
$a = y/2 + m$
$m = a/2$
Random Walk Interpretation

- Imagine a random web surfer
  - At any time \( t \), surfer is on some page \( P \)
  - At time \( t+1 \), the surfer follows an outlink from \( P \) uniformly at random
  - Ends up on some page \( Q \) linked from \( P \)
  - Process repeats indefinitely
- Let \( p(t) \) be a vector whose \( i \)th component is the probability that the surfer is at page \( i \) at time \( t \)
  - \( p(t) \) is a probability distribution on pages

The stationary distribution

- Where is the surfer at time \( t+1 \)?
  - Follows a link uniformly at random
  - \( p(t+1) = Mp(t) \)
- Suppose the random walk reaches a state such that \( p(t+1) = Mp(t) = p(t) \)
  - Then \( p(t) \) is called a stationary distribution for the random walk
- Our rank vector \( r \) satisfies \( r = Mr \)
  - So it is a stationary distribution for the random surfer

Existence and Uniqueness

- A central result from the theory of random walks (aka Markov processes):
  - For graphs that satisfy certain conditions, the stationary distribution is unique and eventually will be reached no matter what the initial probability distribution at time \( t = 0 \).

Spider traps

- A group of pages is a spider trap if there are no links from within the group to outside the group
  - Random surfer gets trapped
- Spider traps violate the conditions needed for the random walk theorem

Microsoft becomes a spider trap

```
<table>
<thead>
<tr>
<th></th>
<th>y</th>
<th>a</th>
<th>m</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>1/2</td>
<td>1/2</td>
<td>0</td>
</tr>
<tr>
<td>a</td>
<td>1/2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>m</td>
<td>0</td>
<td>1/2</td>
<td>1</td>
</tr>
</tbody>
</table>
```

Random teleports

- The Google solution for spider traps
- At each time step, the random surfer has two options:
  - With probability \( \beta \), follow a link at random
  - With probability \( 1-\beta \), jump to some page uniformly at random
  - Common values for \( \beta \) are in the range 0.8 to 0.9
- Surfer will teleport out of spider trap within a few time steps
Matrix formulation

- Suppose there are N pages
  - Consider a page j, with set of outlinks O(j)
  - We have $M_{ij} = 1/|O(j)|$ when j→i and $M_{ij} = 0$ otherwise
  - The random teleport is equivalent to
    - adding a teleport link from j to every other page with probability $(1-\beta)/N$
    - reducing the probability of following each outlink from $1/|O(j)|$ to $\beta/|O(j)|$
  - Equivalent: tax each page a fraction $(1-\beta)$ of its score and redistribute evenly

- Construct the $N \times N$ matrix $A$ as follows
  - $A_{ij} = \beta M_{ij} + (1-\beta)/N$
  - Verify that $A$ is a stochastic matrix
  - The page rank vector $r$ is the principal eigenvector of this matrix
  - satisfying $r = Ar$
  - Equivalently, $r$ is the stationary distribution of the random walk with teleports

Page Rank

- Consider a page j, with set of outlinks O(j)
- We have $M_{ij} = 1/|O(j)|$ when j→i and $M_{ij} = 0$ otherwise
- The random teleport is equivalent to
  - adding a teleport link from j to every other page with probability $(1-\beta)/N$
  - reducing the probability of following each outlink from $1/|O(j)|$ to $\beta/|O(j)|$
  - Equivalent: tax each page a fraction $(1-\beta)$ of its score and redistribute evenly

Previous example with $\beta=0.8$

<table>
<thead>
<tr>
<th></th>
<th>Yahoo</th>
<th>M'soft</th>
<th>Amazon</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>1.00</td>
<td>0.84</td>
<td>0.776</td>
<td>7/11</td>
</tr>
<tr>
<td>a</td>
<td>1.00</td>
<td>0.60</td>
<td>0.536</td>
<td>... 5/11</td>
</tr>
<tr>
<td>m</td>
<td>1.40</td>
<td>1.56</td>
<td>1.688</td>
<td>21/11</td>
</tr>
</tbody>
</table>

Dead ends

- Pages with no outlinks are "dead ends" for the random surfer
  - Nowhere to go on next step

Microsoft becomes a dead end

<table>
<thead>
<tr>
<th></th>
<th>Yahoo</th>
<th>M'soft</th>
<th>Amazon</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>1.00</td>
<td>0.84</td>
<td>0.776</td>
<td>7/11</td>
</tr>
<tr>
<td>a</td>
<td>1.00</td>
<td>0.60</td>
<td>0.536</td>
<td>... 5/11</td>
</tr>
<tr>
<td>m</td>
<td>1.40</td>
<td>1.56</td>
<td>1.688</td>
<td>21/11</td>
</tr>
</tbody>
</table>

Dealing with dead-ends

- Teleport
  - Follow random teleport links with probability 1.0 from dead-ends
  - Adjust matrix accordingly
- Prune and propagate
  - Preprocess the graph to eliminate dead-ends
  - Might require multiple passes
  - Compute page rank on reduced graph
  - Approximate values for deadends by propagating values from reduced graph
Computing page rank

- Key step is matrix-vector multiply: $r_{\text{new}} = Ar_{\text{old}}$
- Easy if we have enough main memory to hold $A$, $r_{\text{old}}$, $r_{\text{new}}$
- Say $N = 1$ billion pages:
  - We need 4 bytes for each entry (say)
  - 2 billion entries for vectors, approx 8GB
  - Matrix $A$ has $N^2$ entries
  - $10^{18}$ is a large number!

Sparse matrix formulation

- Although $A$ is a dense matrix, it is obtained from a sparse matrix $M$
  - 10 links per node, approx 10N entries
- We can restate the page rank equation:
  - $r = (1-\beta)N + \beta Mr$
  - $((1-\beta)/N)N$ is an $N$-vector with all entries $(1-\beta)/N$
- So in each iteration, we need to:
  - Compute $r_{\text{new}} = \beta Mr_{\text{old}}$
  - Add a constant value $(1-\beta)/N$ to each entry in $r_{\text{new}}$

Sparse matrix encoding

- Encode sparse matrix using only nonzero entries:
  - Space proportional roughly to number of links
  - say 10N, or $4\times10\times1$ billion = 40GB
  - still won’t fit in memory, but will fit on disk

Basic Algorithm

- Assume we have enough RAM to fit $r_{\text{new}}$, plus some working memory:
  - Store $r_{\text{old}}$ and matrix $M$ on disk

Basic Algorithm:

- Initialize: $r_{\text{old}} = [1/N]N$
- Iterate:
  - Update: Perform a sequential scan of $M$ and $r_{\text{old}}$ and update $r_{\text{new}}$
  - Write $r_{\text{new}}$ to disk as $r_{\text{old}}$ for next iteration
  - Every few iterations, compute $|r_{\text{new}} - r_{\text{old}}|$ and stop if it is below threshold
  - Need to read in both vectors into memory

Update step

- Initialize all entries of $r_{\text{new}}$ to $(1-\beta)/N$
  - For each page $p$ (out-degree $n$):
    - Read into memory: $p$, $n$, dest$_1$,...,dest$_n$, $r_{\text{old}}(p)$
    - for $j = 1..n$:
      - $r_{\text{new}}(\text{dest}_j) += \beta r_{\text{old}}(p)/n$

Analysis

- In each iteration, we have to:
  - Read $r_{\text{old}}$ and $M$
  - Write $r_{\text{new}}$ back to disk
  - IO Cost = $2|\mathbf{r}| + |\mathbf{M}|$
- What if we had enough memory to fit both $r_{\text{new}}$ and $r_{\text{old}}$?
- What if we could not even fit $r_{\text{new}}$ in memory?
  - 10 billion pages
Block-based update algorithm

<table>
<thead>
<tr>
<th>src</th>
<th>degree</th>
<th>destination</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
<td>0, 1, 3</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>0.5</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3.4</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Analysis of Block Update

- Similar to nested-loop join in databases
  - Break $r^{new}$ into $k$ blocks that fit in memory
  - Scan $M$ and $r^{old}$ once for each block
- $k$ scans of $M$ and $r^{old}$
  - $k(|M| + |r|) + |r| = k|M| + (k+1)|r|
- Can we do better?
- Hint: $M$ is much bigger than $r$ (approx 10-20x), so we must avoid reading it $k$ times per iteration

Block-Stripe Update algorithm

<table>
<thead>
<tr>
<th>src</th>
<th>degree</th>
<th>destination</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
<td>0, 1</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>4</td>
</tr>
</tbody>
</table>

Block-Stripe Analysis

- Break $M$ into stripes
  - Each stripe contains only destination nodes in the corresponding block of $r^{new}$
- Some additional overhead per stripe
  - But usually worth it
- Cost per iteration
  - $|M|(1+\varepsilon) + (k+1)|r|$

Next

- Topic-Specific Page Rank
- Hubs and Authorities
- Spam Detection