Final Projects

• Topic of general interest to the course.
• The idea is to read three or four papers from the literature (references will be provided)
• Implement the application on the cluster you build
• Synthesize them in terms of a report (~20 pages)
• Present your report to class (~20 mins)
• New ideas and extensions are welcome, as well as implementation prototype if needed.
Google query attributes
- 150M queries/day (2000/second)
- 100 countries
- 8.0B documents in the index

Data centers
- 100,000 Linux systems in data centers around the world
  - 15 TFlop/s and 1000 TB total capability
  - 40-80 1U/2U servers/cabinet
  - 100 MB Ethernet switches/cabinet with gigabit Ethernet uplink
- growth from 4,000 systems (June 2000)
  - 18M queries then

Performance and operation
- simple reissue of failed commands to new servers
- no performance debugging
  - problems are not reproducible

Eigenvalue problem; $Ax = \lambda x$
$n = 8 \times 10^9$
(see: MathWorks Cleve's Corner)
The matrix is the transition probability matrix of the Markov chain; $Ax = x$

Source: Monika Henzinger, Google & Cleve Moler
Imagine a library containing 1 trillion documents but with no centralized organization and no librarians.

In addition, anyone may add a document at any time without telling anyone.

You may feel sure that one of the documents contained in the collection has a piece of information that is vitally important to you, and, being impatient like most of us, you'd like to find it in a matter of seconds.

How would you go about doing it?

Posed in this way, the problem seems impossible.
Every Page has a Rank

- PageRank is determined entirely by the link structure of the Web.
- It is recomputed about once a month and does not involve any of the actual content of Web pages or of any individual query.
- Then, for any particular query, Google finds the pages on the Web that match that query and lists those pages in the order of their PageRank.
Imagine surfing the Web, going from page to page by randomly choosing an outgoing link from one page to get to the next.

This can lead to dead ends at pages with no outgoing links, or cycles around cliques of interconnected pages.

So, a certain fraction of the time, simply choose a random page from anywhere on the Web.

This theoretical random walk of the Web is a Markov chain or Markov process.

The limiting probability that a dedicated random surfer visits any particular page is its PageRank.

A page has high rank if it has links to and from other pages with high rank.
The Matrix

- Let $W$ be the set of Web pages that can be reached by following a chain of hyperlinks starting from a page at Google and let $n$ be the number of pages in $W$.
- The set $W$ actually varies with time, but in July 2008 there were 1 trillion pages.
- Let $G$ be the $n$-by-$n$ connectivity matrix of $W$, that is, $g_{i,j}$ is 1 if there is a hyperlink from page $i$ to page $j$ and 0 otherwise.
- The matrix $G$ is huge, but very sparse; its number of nonzeros is the total number of hyperlinks in the pages in $W$. 
The Matrix

- Let $c_j$ and $r_i$ be the column and row sums of $G$.
- $c_j = \Sigma_i g_{i,j}$, $r_i = \Sigma_j g_{i,j}$
- The quantities $c_k$ and $r_k$ are the indegree and outdegree of the $k$-th page.
- Let $p$ be the fraction of time that the random walk follows a link.
- Google usually takes $p = 0.85$. Then $1-p$ is the fraction of time that an arbitrary page is chosen.
- Let $A$ be the $n$-by-$n$ matrix whose elements are
  - $a_{i,j} = p g_{i,j} / c_j + \delta$, where $\delta = (1-p) / n$ when $c_j \neq 0$
  - $a_{i,j} = 1/n$ when $c_j = 0$ (more on this later)
The matrix $A$ is not sparse, but it is a rank one modification of a sparse matrix. Most of the elements of $A$ are equal to the small constant $\delta$.

When $n = 2.7 \cdot 10^9$, $\delta = 5.5 \cdot 10^{-11}$.

The matrix is the transition probability matrix of the Markov chain.

Its elements are all strictly between zero and one and its column sums are all equal to one.
Eigenvalue Problem

- An important result in matrix theory, the Perron-Frobenius Theorem, applies to such matrices.
- It tells us that the largest eigenvalue of $A$ is equal to one and that the corresponding eigenvector, which satisfies the equation $x = Ax$, exists and is unique to within a scaling factor. When this scaling factor is chosen so that $\Sigma_i x_i = 1$
- then $x$ is the state vector of the Markov chain. The elements of $x$ are Google's PageRank.
Example
Link Matrix

\[
\begin{bmatrix}
0 & \frac{1}{2} & 0 & 0 & \frac{1}{2} & 0 \\
\frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 & 0 \\
0 & \frac{1}{4} & 0 & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \\
0 & \frac{1}{3} & \frac{1}{3} & 0 & 0 & \frac{1}{3} \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & \frac{1}{2} & 0 & \frac{1}{2} & 0
\end{bmatrix}
\]
Definitions

- **Markov chain**
  - The conditional probability of each future state depends only on the present state

- **Markov matrix**
  - Transition matrix of a Markov chain
Transition Matrix

From our earlier mini-web:

\[
H = \begin{bmatrix}
0 & \frac{1}{2} & 0 & 0 & \frac{1}{2} & 0 \\
\frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 & 0 \\
0 & \frac{1}{4} & 0 & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \\
0 & \frac{1}{3} & \frac{1}{3} & 0 & 0 & \frac{1}{3} \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & \frac{1}{2} & 0 & \frac{1}{2} & 0
\end{bmatrix}
\]
Markov Matrix Properties

- **Row-stochastic**
- **Stationary vector gives long-term probability of each state**
- **All eigenvalues $\lambda \leq 1$**
\[
\begin{bmatrix}
0 & \frac{1}{2} & 0 & 0 & \frac{1}{2} & 0 \\
\frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 & 0 \\
0 & \frac{1}{4} & 0 & \frac{1}{4} & \frac{1}{4} & \frac{1}{3} \\
0 & \frac{1}{3} & \frac{1}{3} & 0 & 0 & \frac{1}{3} \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & \frac{1}{2} & 0 & \frac{1}{2} & 0 \\
\end{bmatrix}
\]

not row-stochastic
Define a vector $\mathbf{a}$ such that:

$$a_i = \begin{cases} 
1 & ; \text{row } i \text{ corresponds to a dangling node} \\
0 & ; \text{otherwise}
\end{cases}$$

Then we obtain a row-stochastic matrix:

$$S = H + \mathbf{a}\mathbf{u}^T$$
- We have our matrix and now we want to find the eigenvector corresponding to the eigenvalue at 1.
- Eigenvalue problem is: \(Ax = \lambda x\) where \(\lambda = 1\).
- Solve \(Ax = x\); \((A-I)x = 0\);
- Use Power Method:
  \(Ax_i = x_{i+1}\)
- When it converges \(x\) is the PageRank.

\[
\begin{bmatrix}
0 & \frac{1}{2} & 0 & 0 & \frac{1}{2} & 0 \\
\frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 & 0 \\
0 & \frac{1}{4} & 0 & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \\
0 & \frac{1}{3} & \frac{1}{3} & 0 & 0 & \frac{1}{3} \\
\frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} \\
0 & 0 & \frac{1}{2} & 0 & \frac{1}{2} & 0 \\
\end{bmatrix}
\]
Now we are ready to search

- So we have the PageRank vector
- Need an inverted index of the words on the web.
- List of words to search
Inverted Index

- We need a way to map search words to web pages.
- Inverted Index can do that.
- An inverted index maps from content, such as words or numbers, to its locations or web pages.
- Think big...
- Form a list of all the words in all the web pages
- For each word point to the webpages that have the word.
Inverted index

Document 1
The bright blue butterfly hangs on the breeze.

Document 2
It's best to forget the great sky and to retire from every wind.

Document 3
Under blue sky, in bright sunlight, one need not search around.

Stopword list
a and around every for from in is it not on one the to under

Inverted index

<table>
<thead>
<tr>
<th>ID</th>
<th>Term</th>
<th>Document</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>best</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>blue</td>
<td>1, 3</td>
</tr>
<tr>
<td>3</td>
<td>bright</td>
<td>1, 3</td>
</tr>
<tr>
<td>4</td>
<td>butterfly</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>breeze</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>forget</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>great</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>hangs</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>need</td>
<td>3</td>
</tr>
<tr>
<td>10</td>
<td>retire</td>
<td>2</td>
</tr>
<tr>
<td>11</td>
<td>search</td>
<td>3</td>
</tr>
<tr>
<td>12</td>
<td>sky</td>
<td>2, 3</td>
</tr>
<tr>
<td>13</td>
<td>wind</td>
<td>2</td>
</tr>
</tbody>
</table>
Inverted index

- When a request comes in look up the pages that have the words.
- Take the intersection of pages that contain the words.
- Use the PageRank vector to order the pages that are returned.