Google!

Google

Query: "Java"

The Source for **Java** Technology
The Source for **Java** Technology. The **Java** 2 Platform **...** Get **Java**. Highlights November 4, 2003 Play Ball! Tendu's **Java** software applications **...**
Description: Sun's home for **Java**. Offers Windows, Solaris, and Linux **Java** Development Kits (JDKs), extensions,...
Category: Computers > Programming > Languages > Java
java.sun.com/ - 46k - 9 Nov 2003 - Cached - Similar pages

The Java Tutorial
... Developer Services. **Java** BluePrints. **Java** Tutorial. **...** Your First Cup of **Java**: Detailed instructions to help you run your first program: UNIX, Microsoft Windows, Mac. **...**
Description: On-line version of book from Addison-Wesley.
Category: Computers > Programming > ... > Tutorials
java.sun.com/docs/books/tutorial/ - 40k - 9 Nov 2003 - Cached - Similar pages
[More results from java.sun.com]

Java(TM) Boutique - Free **Java** Applets, Games, Programming ...
The **Java** Boutique is a collection of **java** applets, games, scripts, and tutorials. Learn programming and download free **java** applets and source code. **...**
Description: Collection of **Java** applets, script information and tutorials.
Category: Kids and Teens > Computers > ... > CGI and Programming > Java
javaboutique.internet.com/ - 50k - 9 Nov 2003 - Cached - Similar pages

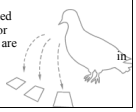
Pigeon-ranking system

The technology behind Google's great results

As a Google user, you're familiar with the speed and accuracy of a Google search. How exactly does Google manage to find the right results for every query as quickly as it does? The heart of Google's search technology is PigeonRank™, a system for ranking web pages developed by Google founders Larry Page and Sergey Brin at Stanford University.

Why Google's patented PigeonRank™ works so well

PigeonRank's success relies primarily on the superior trainability of the domestic pigeon (Columba livia) and its unique capacity to recognize objects regardless of spatial orientation. The common gray pigeon can easily distinguish among items displaying only the minutest differences, an ability that enables it to select relevant web sites from among thousands of similar pages. By collecting flocks of pigeons in dense clusters, Google is able to process search queries at speeds superior to traditional search engines, which typically rely on birds of prey, brooding hens or slow-moving waterfowl to do their relevance rankings. When a search query is submitted to Google, it is routed to a data coop where monitors flash result pages at blazing speeds. When a relevant result is observed by one of the pigeons in the cluster, it strikes a rubber-coated steel bar with its beak, which assigns the page a PigeonRank value of one. For each peck, the PigeonRank increases. Those pages receiving the most pecks, are returned at the top of the user's results page with the other results displayed in pecking order.



- **Read more at http://www.google.com/technology/pigeonrank.html**

How google really works

1. Web crawling:
   - Use depth-first or breadth-first search to:
     Learn the structure (vertices+edges) of the graph
     Build an index of the web:
     Use a hash table to store pairs (word, list-of-sites)

index: | ... \| "jack" \| "java" \| "jet"\| .... |

http://www.jackdaniels.com http://www.jackinthebox.com ...	http://www.sun.com http://www.mcb.mcgill.ca ...

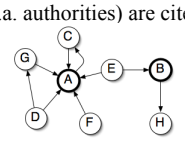
```
for each web site S do
    for each word w in S do
        index.get(w).addLast(S)
```

Page ranking system

- Idea #1: Web pages reported should contain the query words.
  - Easy: Simply use the index
  - Variants:
    - Better to have several occurrences of the words in the query. SEX SEX SEX SEX SEX SEX SEX SEX
    - Allow synonyms
    - Use context to determine meaning of words:
      Query: "java error"
      Bad site: "It is an error to serve java coffee with milk"

Exploiting graph structure

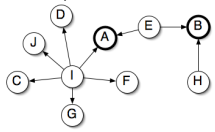
- Idea #2:
  - Website authors know good sites of a particular domain and have links to them
  - Good sites (a.k.a. authorities) are cites by many other sites



  - Prefer websites with large in-degree

Exploiting graph structure

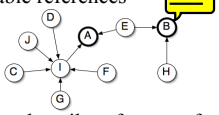
- Idea #3:
  - Websites that link to a large number of other sites (a.k.a hubs) are less valuable references



  - Weight less heavily references from sites that have large out-degree

Exploiting graph structure

- Idea #4:
  - Websites that are themselves authorities are more valuable references



  - Weight more heavily references from sites that have high page-rank (even if they don't contain words from the query)

Putting it all together

- How to incorporate ideas #2, #3, #4?
- The page-rank PR(v) of vertex v describes how authoritative this site v is
  - based only on the graph structure, not on the actual query)
- High page-rank is good
- To answer a query:
  - Find all sites that contain the words of the query
  - Sort them in decreasing order of page-rank

Computing page-ranks

- Let PR(v) be the page-rank of vertex v
- Let C(v) be the out-degree of vertex v
- Let $w_1, w_2, ..., w_k$ be the sites that have links to v

$$PR(v) = \frac{PR(w_1)}{C(w_1)} + \frac{PR(w_2)}{C(w_2)} + ... + \frac{PR(w_k)}{C(w_k)}$$

- For technical reasons, add a damping factor d:

$$PR(v) = (1-d) + d * \left(\frac{PR(w_1)}{C(w_1)} + \frac{PR(w_2)}{C(w_2)} + ... + \frac{PR(w_k)}{C(w_k)} \right)$$

Solving for PR(v)

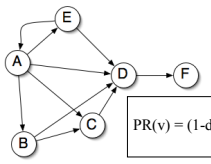
- PR(v) is expressed as a function of $PR(w_1),...,PR(w_k)$
- Let $v_1,..., v_n$ be all the vertices of the internet graph
- We have a system of linear equations with unknowns $PR(v_1)$, ... $PR(v_n)$
- We could use linear algebra to solve for $PR(v_1)$, ... $PR(v_n)$
  - Gaussian elimination. Problem: runs in $O(n^3)$
- Instead we use a simple numerical approximation method...

Fixed-point iterative solution

- To solve a system of equations

$$x_1 = f_1(x_1, x_2, ..., x_n)$$
$$x_2 = f_2(x_1, x_2, ..., x_n)$$
$$...$$
$$x_n = f_n(x_1, x_2, ..., x_n)$$

Repeat
  for i = 1 to n do $x_i = 1$
  for i = 1 to n do $t_i = f_i(x_1, x_2, ..., x_n)$
  for i = 1 to n do $x_i = t_i$
until convergence



$$PR(v) = (1-d) + d * \left(\frac{PR(w_1)}{C(w_1)} + \frac{PR(w_2)}{C(w_2)} + \ldots + \frac{PR(w_k)}{C(w_k)} \right)$$

with d=1/2, we get

0-th iteration: PR(A)=PR(B)=PR(C)=PR(D)=PR(E)=PR(F)=1

1-st iteration (using the val m iteration 0)

PR(A) = 1/2 + 1/2 * (PR(E) / 2) = 3/4,
PR(B) = 1/2 + 1/2 * ( ) =
PR(C) = 1/2 + 1/2 * ( ) =
PR(D) = 1/2+1/2 * ( )=
PR(E) = 1/2 + 1/2 * ( ) =
PR(F) = 1/2 + 1/2* ( ) =

1-st iteration (just repeating last slide's result)

PR(A)=3/4, PR(B)=5/8, PR(C)=7/8, PR(D)=13/8, PR(E)=5/8, PR(F)=1

2-nd iteration (using values of the previous iteration)

PR(A) = 1/2 + 1/2 * (PR(E) / 2) = 21/32
PR(B) = 1/2 + 1/2 * (PR(A) / 4) = 19/32
PR(C) = 1/2 + 1/2 * (PR(A) / 4 + PR(B) / 2) = 3/4
PR(D) = 1/2+1/2*(PR(A)/4 + PR(B)/2 + PR(C)/1 +PR(E)/2)= 43/32
PR(E) = 1/2 + 1/2 * (PR(A) / 4) = 19/32
PR(F) = 1/2 + 1/2* (PR(D) / 1) = 42/32

After 10 iterations, all PR() are stabilized to the 10th decimal:

PR(A) = 0.645..., PR(B) = 0.581..., PR(C) = 0.726...,
PR(D) = 1.234..., PR(E) = 0.581..., PR(F) = 1.117...

Ordering of the pages based on page-rank: D, F, C, A, B, E