Perspektiverende Datalogi
Internetalgoritmer

Gerth Stølting Brodal

dPersp - Internetalgoritmer

Ugens program

Mandag 6/10	12.15-14.00	Forelæsning: Internetalgoritmer - Google's PageRank *Gerth Stølting Brodal (Aud E)*
Tirsdag 7/10	9.15-12.00	Øvelser
	12.15-13.00	Forelæsning: MapReduce *Gerth Stølting Brodal (PBA Auditoriet)*
	13.15-16.00	Øvelser
Onsdag 8/10	14.15-16.00	Historisk perspektiv: Programmeringssprog Den afsluttende opgave over "As We May Think" stilles Kort "her-og-nu" evaluering af dPersp14 (passende skriveredskab bedes medbragt) *Erik Meineche Schmidt (PBA Auditoret)*

The Anatomy of a Large-Scale Hypertextual Web Search Engine

Sergey Brin and Lawrence Page

Computer Science Department,
Stanford University, Stanford, CA 94305, USA
sergey@cs.stanford.edu and page@cs.stanford.edu

Abstract

In this paper, we present Google, a prototype of a large-scale search engine which makes heavy use of the structure present in hypertext. Google is designed to crawl and index the Web efficiently and produce much more satisfying search results than existing systems. The prototype with a full text and hyperlink database of at least 24 million pages is available at http://google.stanford.edu/
To engineer a search engine is a challenging task. Search engines index tens to hundreds of millions of web pages involving a comparable number of distinct terms. They answer tens of millions of queries every day. Despite the importance of large-scale search engines on the web, very little academic research has been done on them. Furthermore, due to rapid advance in technology and web proliferation, creating a web search engine today is very different from three years ago. This paper provides an in-depth description of our large-scale web search engine -- the first such detailed public description we know of to date. Apart from the problems of scaling traditional search techniques to data of this magnitude, there are new technical challenges involved with using the additional information present in hypertext to produce better search results. This paper addresses this question of how to build a practical large-scale system which can exploit the additional information present in hypertext. Also we look at the problem of how to effectively deal with uncontrolled hypertext collections where anyone can publish anything they want.

Keywords

World Wide Web, Search Engines, Information Retrieval, PageRank, Google

1. Introduction

(Note: There are two versions of this paper -- a longer full version and a shorter printed version. The full version is available on the web and the conference CD-ROM.)
The web creates new challenges for information retrieval. The amount of information on the web is growing rapidly, as well as the number of new users inexperienced in the art of web research. People are likely to surf the web using its link graph, often starting with high quality human maintained indices such as Yahoo! or with search engines. Human maintained lists cover popular topics effectively but are subjective, expensive to build and maintain, slow to improve, and cannot cover all esoteric topics. Automated search engines that rely on keyword matching usually return too many low quality matches. To make matters worse, some advertisers attempt to gain people's attention by taking measures meant to mislead automated search engines. We have built a large-scale search engine which addresses many of the problems of existing systems. It makes especially heavy use of the additional structure present in hypertext to provide much higher quality search results. We chose our system name, Google, because it is a common spelling of googol, or 10^{100} and fits well with our goal of building very large-scale search

Brin, S. and Page, L. (1998)
The Anatomy of a Large-Scale Hypertextual Web Search Engine.
In: Seventh International World-Wide Web Conference (WWW 1998)

MapReduce: Simplified Data Processing on Large Clusters

Jeffrey Dean and Sanjay Ghemawat

jeff@google.com, sanjay@google.com

Google, Inc.

Abstract

MapReduce is a programming model and an associated implementation for processing and generating large data sets. Users specify a *map* function that processes a key/value pair to generate a set of intermediate key/value pairs, and a *reduce* function that merges all intermediate values associated with the same intermediate key. Many real world tasks are expressible in this model, as shown in the paper.

Programs written in this functional style are automatically parallelized and executed on a large cluster of commodity machines. The run-time system takes care of the details of partitioning the input data, scheduling the program's execution across a set of machines, handling machine failures, and managing the required inter-machine communication. This allows programmers without any experience with parallel and distributed systems to easily utilize the resources of a large distributed system.

Our implementation of MapReduce runs on a large cluster of commodity machines and is highly scalable: a typical MapReduce computation processes many terabytes of data on thousands of machines. Programmers find the system easy to use: hundreds of MapReduce programs have been implemented and upwards of one thousand MapReduce jobs are executed on Google's clusters every day.

1 Introduction

Over the past five years, the authors and many others at Google have implemented hundreds of special-purpose computations that process large amounts of raw data, such as crawled documents, web request logs, etc., to compute various kinds of derived data, such as inverted indices, various representations of the graph structure of web documents, summaries of the number of pages crawled per host, the set of most frequent queries in a given day, etc. Most such computations are conceptually straightforward. However, the input data is usually large and the computations have to be distributed across hundreds or thousands of machines in order to finish in a reasonable amount of time. The issues of how to parallelize the computation, distribute the data, and handle failures conspire to obscure the original simple computation with large amounts of complex code to deal with these issues.

As a reaction to this complexity, we designed a new abstraction that allows us to express the simple computations we were trying to perform but hides the messy details of parallelization, fault-tolerance, data distribution and load balancing in a library. Our abstraction is inspired by the *map* and *reduce* primitives present in Lisp and many other functional languages. We realized that most of our computations involved applying a *map* operation to each logical "record" in our input in order to compute a set of intermediate key/value pairs, and then applying a *reduce* operation to all the values that shared the same key, in order to combine the derived data appropriately. Our use of a functional model with user-specified map and reduce operations allows us to parallelize large computations easily and to use re-execution as the primary mechanism for fault tolerance.

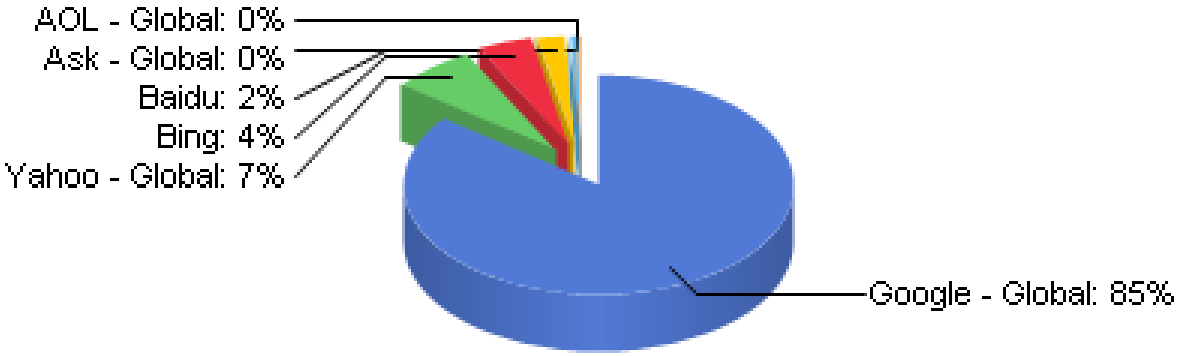
The major contributions of this work are a simple and powerful interface that enables automatic parallelization and distribution of large-scale computations, combined with an implementation of this interface that achieves high performance on large clusters of commodity PCs.

Section 2 describes the basic programming model and gives several examples. Section 3 describes an implementation of the MapReduce interface tailored towards our cluster-based computing environment. Section 4 describes several refinements of the programming model that we have found useful. Section 5 has performance measurements of our implementation for a variety of tasks. Section 6 explores the use of MapReduce within Google including our experiences in using it as the basis

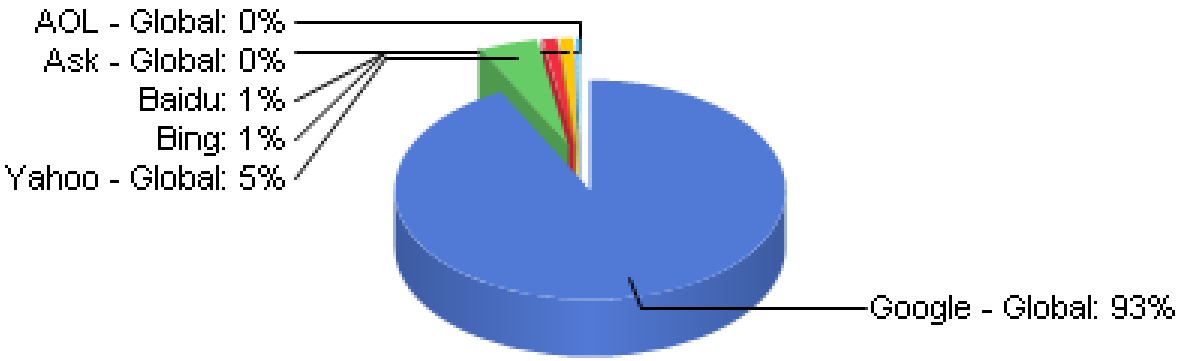
Dean, F. and Ghemawat, S. (2004) *MapReduce: Simplified Data Processing on Large Clusters*. In: Sixth Symposium on Operating System Design and Implementation (OSDI 2004): 137-150

Internetsøgemaskiner

(29. september 2012)



Desktop

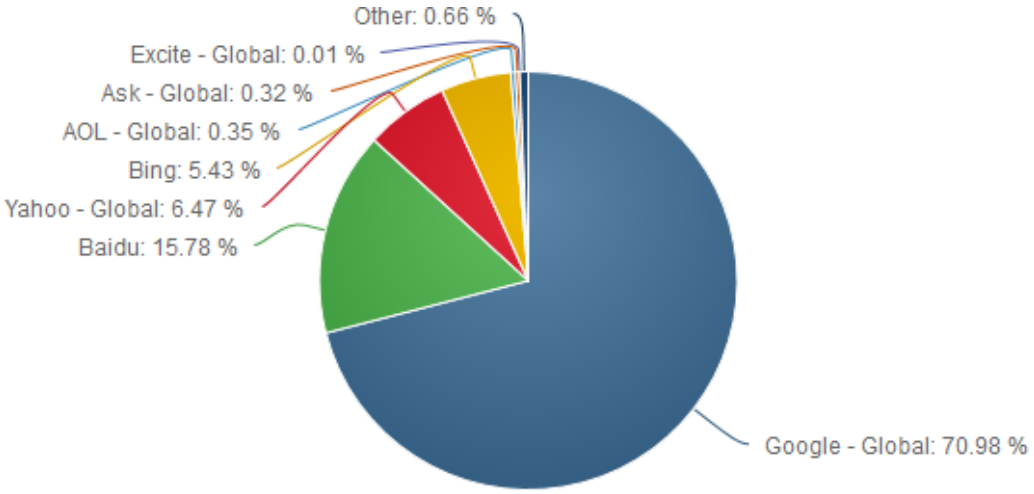


Mobile enheder

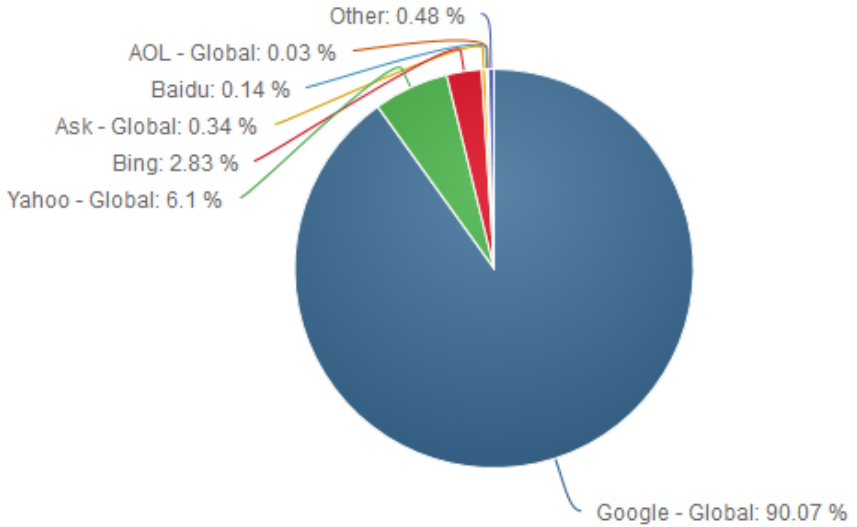
marketshare.hitslink.com

Internetsøgemaskiner

(30. september 2013)



Desktop

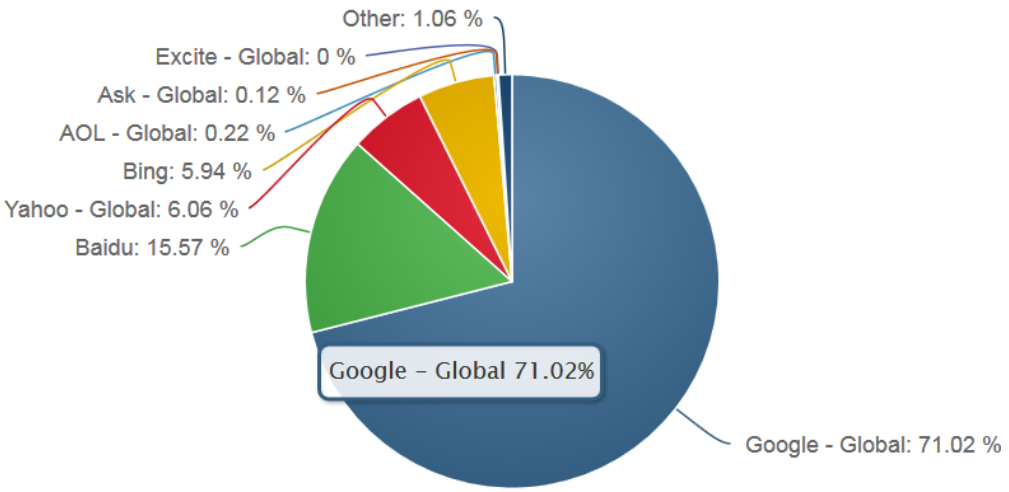


Mobile enheder

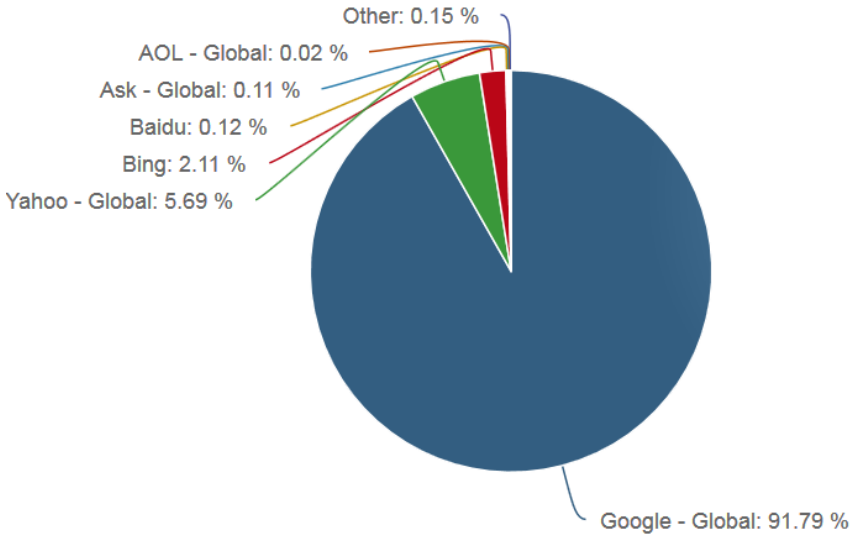
marketshare.hitslink.com

Internetsøgemaskiner

(5. oktober 2014)



Desktop



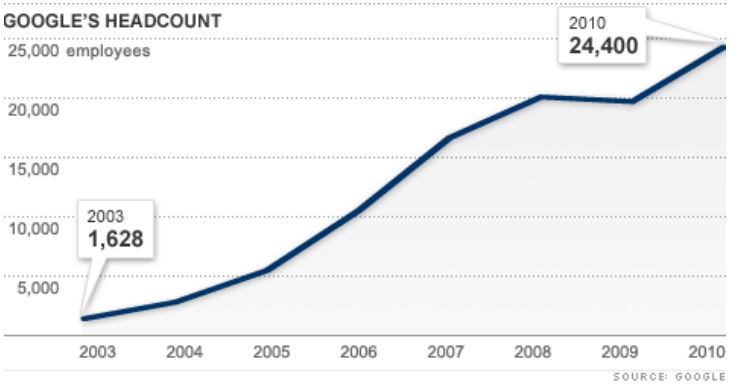
Mobile enheder

marketshare.hitslink.com

Google™

google ≈ googol = 10^{100}

- Startet i 1995 som forskningsprojekt ved Stanford Universitet af ph.d. studerende **Larry Page og Sergey Brin**
- Privat firma grundlagt 1998
- Hovedsæde i Silicon Valley

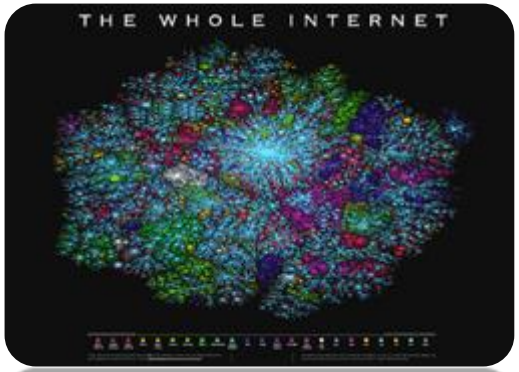


www.google.com/about/company/products/


Google
Baidu 百度
msn
HOTBOT
altavista
alltheweb
find it all
Jubii
-or not to be
AOL Search
LYCOS
YAHOO! SEARCH
excite
bing
Ask
.com

Internettet - Historiske Perspektiv

- 1969 Første ARPANET forbindelser - starten på internettet
- 1971 Første email, FTP
- 1990 HTML sproget defineres
- 1991-1993 få kender til HTML
- 1993 Mosaic web-browser
- 1993 Lycos
- 1994 WebCrawler
- 1995 Yahoo!, Altavista - *mange sider*
- **1998 Google -** *mange sider og* ***god ranking***
- 2004 Facebook
- 2005 YouTube
- ....



World Wide Web

- **Internettet** = stor mængde **ustruktureret** information.
- Hvordan finder man relevant info? **Søgemaskiner**!

Firefox

"Nobodys Home" by Tina Dickow on ... | unf - Google-søgning

http://www.google.dk/search?q=unf&ie=utf-8&oe=utf-8&aq=t&rls=org.mozilla:en-US:official&client=fire

unf

Nettet Billeder Kort Oversæt Scholar Blogs Gmail mere ▾ Søgeindstillinger | Log ind

Google

unf Søg

Ca. 6.480.000 resultater (0,09 sekunder) Avanceret søgning

Alle
Billeder
Videoer
Mere

Aarhus
Skift placering

Nettet
Sider på dansk
Sider fra Danmark
Oversatte udenlandske sider

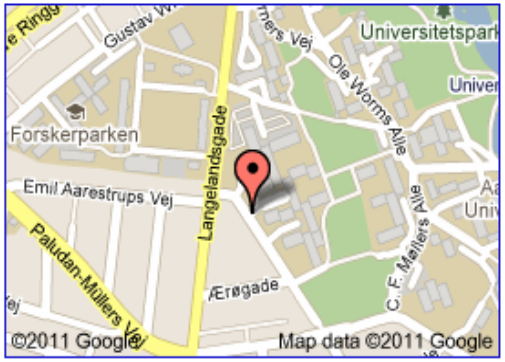
Ethvert tidsinterval
Seneste
De seneste 24 timer
De seneste 2 dage
Den seneste uge
Den seneste måned
Det seneste år
Tilpasset interval ...

Flere værktøjer

UNF Danmark
Tilbyder foredrag, studiebesøg og nyheder indenfor alle grene af naturvidenskaben.
www.unf.dk/ - Cached - Lignende

København Om UNF
Odense Sciencecamps
Århus Medlemskab
Aalborg Matematik

Flere resultater fra unf.dk »



UNF Århus
stedside
Ny Munkegade
8000 Aarhus Municipality
8942 3345
Bus: Langelandsgade/Kaserneboulevar
Hent anvisninger

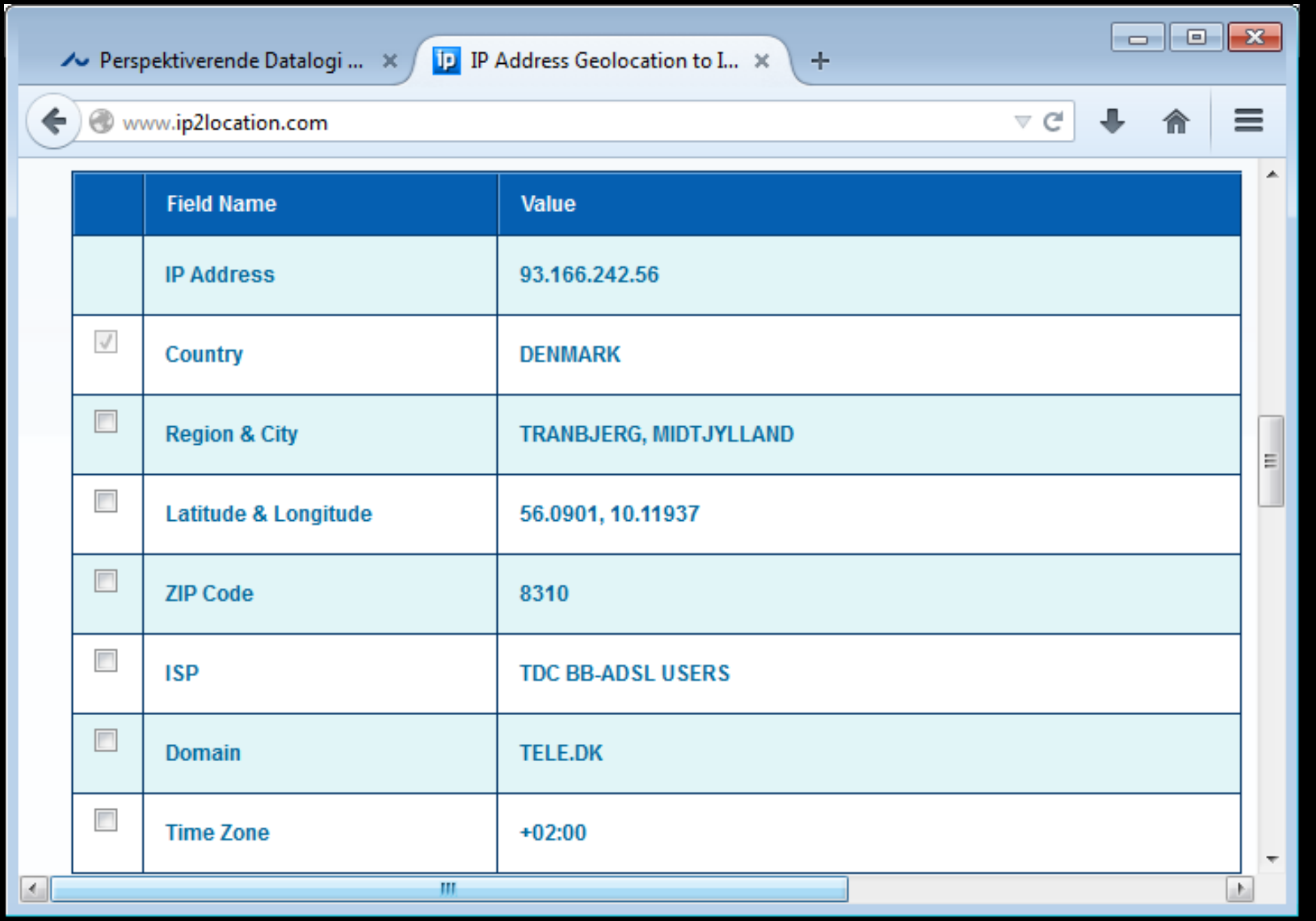
1 anmeldelse

UNF København
Hvad skal **UNF** KBH holde foredrag om d. 3. marts? [Læs beskrivelse] ...
kbh.unf.dk/ - Cached - Lignende

UNF Odense
Bag alle disse spørgsmål gemmer der sig en naturvidenskabelige forklaring ...
odense.unf.dk/ - Cached - Lignende

UNF Århus
Går du på gymnasiet eller HTX er der nu to muligheder for at komme billigt ...
aarhus.unf.dk/ - Cached - Lignende

Vis flere resultater fra unf.dk



	Field Name	Value
	IP Address	93.166.242.56
☑	Country	DENMARK
☐	Region & City	TRANBJERG, MIDTJYLLAND
☐	Latitude & Longitude	56.0901, 10.11937
☐	ZIP Code	8310
☐	ISP	TDC BB-ADSL USERS
☐	Domain	TELE.DK
☐	Time Zone	+02:00

Moderne Søgemaskiner

Imponerende performance

- Søger i 10^{10} sider
- Svartider 0,1 sekund
- 1000 brugere i sekundet
- Finder relevante sider

I'm Feeling Lucky

Krav til Søgemaskiner

- Dynamiske websider:

  Nyheder, Twitter, Facebook, ...
- **Personlig ranking**

  Baseret på hidtige besøgte websider, gmail, social netværk, geografisk placering, ...

Google™ (2004)

- +8.000.000.000 web sider (+20 TB)
- PageRank: +3.000.000.000 sider og +20.000.000.000 links
- +2 Terabyte index, opdateres en gang om måneden
- +2.000.000 termer i indeks
- +150.000.000 søgninger om dagen (2000 i sekundet)
- +200 filtyper: HTML, Microsoft Office, PDF, PostScript, WordPerfect, Lotus ...
- +28 sprog

Google™ (2004)

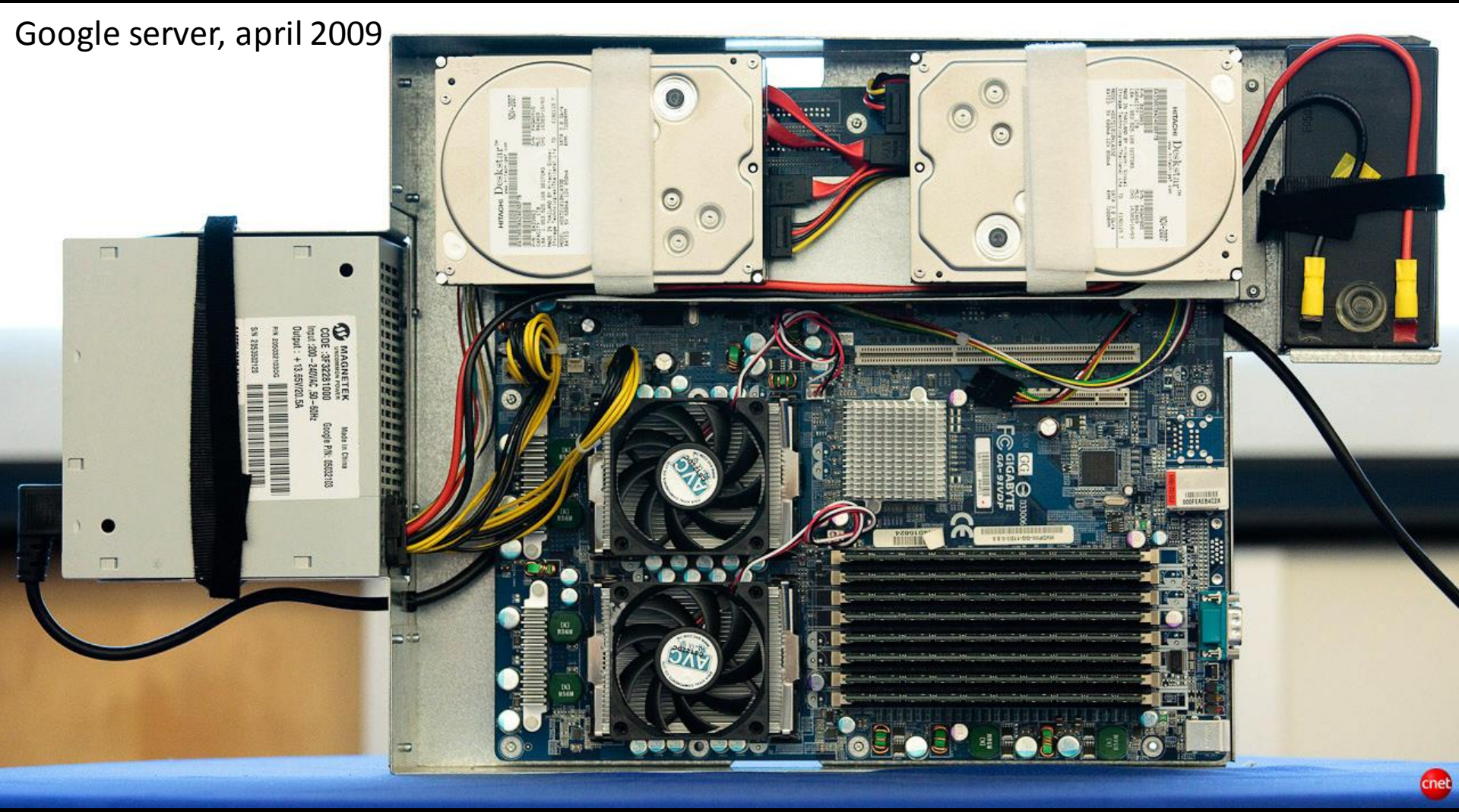
- Cluster af +10.000 Intel servere med Linux
  - Single-processor
  - 256MB–1GB RAM
  - 2 IDE diske med 20-40Gb
- Fejl-tolerance: Redundans
- Hastighed: Load-balancing


Bryce Crane Co.
EXIT


AU 14
AU-14


www.google.dk/about/datacenters/

Google server, april 2009



Google™ Datacentre

www.google.com/about/datacenters/inside/locations/index.html

En søgemaskines dele

Indsamling af data

- Webcrawling (gennemløb af internet)

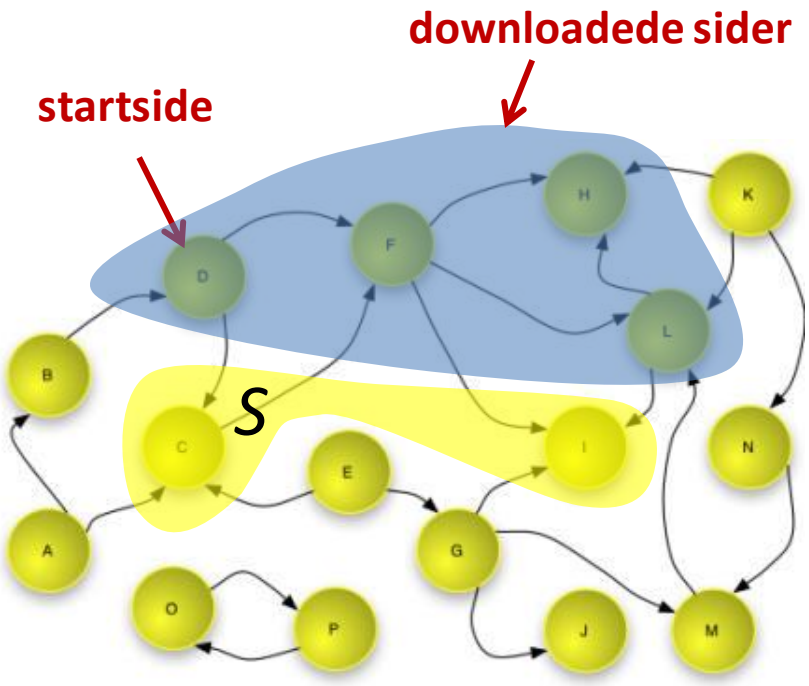
Indeksering data

- Parsning af dokumenter
- Leksikon: indeks (ordbog) over alle ord mødt
- Inverteret fil: for alle ord i leksikon, angiv i hvilke dokumenter de findes

Søgning i data

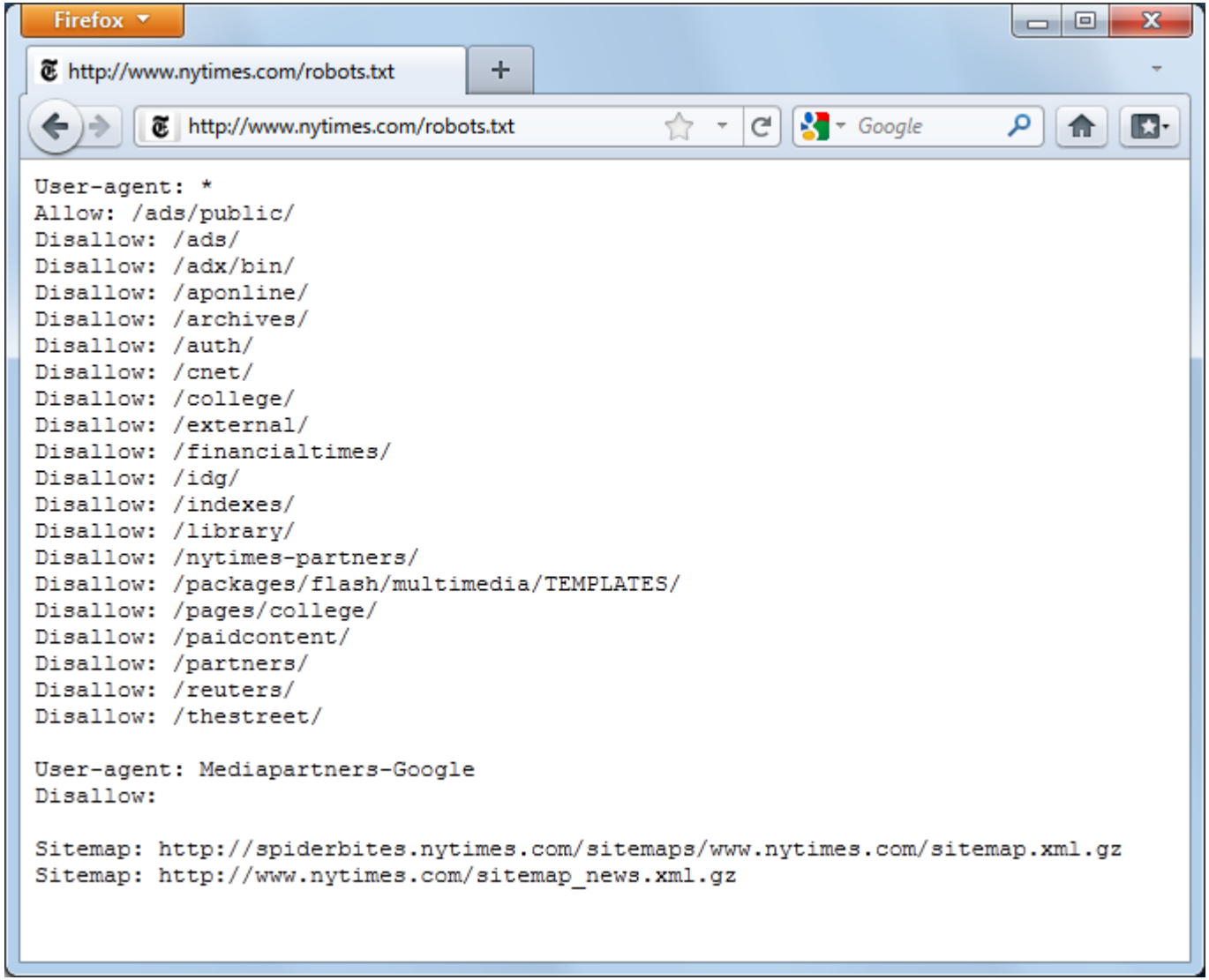
- Find alle dokumenter med søgeordene
- Rank dokumenterne

Webcrawling = Grafgennemløb



S = {startside}
repeat
  fjern en side *s* fra *S*
  parse *s* og find alle links (*s*, *v*)
  foreach (*s*, *v*)
    if *v* ikke besøgt før
      indsæt *v* i *S*

robots.txt @ The New York Times




```
User-agent: *
Allow: /ads/public/
Disallow: /ads/
Disallow: /adx/bin/
Disallow: /aponline/
Disallow: /archives/
Disallow: /auth/
Disallow: /cnet/
Disallow: /college/
Disallow: /external/
Disallow: /financialtimes/
Disallow: /idg/
Disallow: /indexes/
Disallow: /library/
Disallow: /nytimes-partners/
Disallow: /packages/flash/multimedia/TEMPLATES/
Disallow: /pages/college/
Disallow: /paidcontent/
Disallow: /partners/
Disallow: /reuters/
Disallow: /thestreet/

User-agent: Mediapartners-Google
Disallow:

Sitemap: http://spiderbites.nytimes.com/sitemaps/www.nytimes.com/sitemap.xml.gz
Sitemap: http://www.nytimes.com/sitemap_news.xml.gz
```

Robusthed

- Normalisering af URLer
- Parsning af malformet HTML
- Mange filtyper
- Forkert content-type fra server
- Forkert HTTP response code fra server
- Enorme filer
- Uendelige URL-løkker (crawler traps)
- …

Vær konservativ – opgiv at finde alt
Crawling kan tage måneder

Designovervejelser - Crawling

- Startpunkt (initial *S*)
- Crawl-strategi (valg af *s*)
- Mærkning af besøgte sider
- Robusthed
- Ressourceforbrug (egne og andres ressourcer)
- Opdatering: Kontinuert vs. periodisk crawling

```
S = {startside}
repeat
    fjern en side s fra S
    parse s og find alle links (s, v)
    foreach (s, v)
        if v ikke besøgt før
            indsæt v i S
```

Output: **DB med besøgte dokumenter**
DB med links i disse (kanterne i Internetgrafen)
DB med DokumentID–URL mapning

Resourceforbrug

- Egne resourcer
  - Båndbredde (global request rate)
  - Lagerplads (brug kompakte repræsentationer)
  - Distribuér på flere maskiner
- Andres resourcer (politeness)
  - Båndbredde (lokal request rate); tommelfingerregel: 30 sekunder mellem request til samme site.
- Robots Exclusion Protocol (www.robotstxt.org)
- Giv kontakt info i HTTP-request

Erfaringer ang. Effektivitet

- Brug caching (DNS opslag, robots.txt filer, senest mødte URL'er)
- Flaskehals er ofte disk I/O under tilgang til datastrukturerne
- CPU cycler er ikke flaskehals
- En tunet crawler (på een eller få maskiner) kan crawle 200-400 sider/sek 35 mio sider/dag

Indeksering af dokumenter

- Preprocessér en dokumentsamling så dokumenter med et givet søgeord kan blive returneret hurtigt

Input: dokumentsamling

Output: søgestruktur

Indeksering: Inverteret fil + leksikon

- **Inverteret fil** = for hvert ord *w* en liste af dokumenter indeholdende *w*
- **Leksikon** = ordbog over alle forekommende ord (nøgle = ord, værdi = pointer til liste i inverteret fil + evt. ekstra info for ordet, fx længde af listen)

For en milliard dokumenter:

Inverteret fil: totale antal ord 100 mia DISK

Leksikon: antal forskellige ord 2 mio RAM

Inverteret Fil

- Simpel (forekomst af ord i dokument):

  ord1: DocID, DocID, DocID

  ord2: DocID, DocID

  ord3: DocID, DocID, DocID, DocID, DocID, ...

  ...

- Detaljeret (*alle forekomster af ord i dokument):*

  ord1: DocID, Position, Position, DocID, Position ...

  ...

- Endnu mere detaljeret:

  Forekomst annoteret med info
  (heading, boldface, anker text,. . .)
  Kan bruges under ranking

Bygning af index

foreach dokument **D** i samlingen
    Parse **D** og identificér ord
    foreach ord w
        Udskriv (DocID(**D**), w)
        if w ikke i leksikon
            indsæt w i leksikon

(1, 2), (1, 37), … , (1, 123) , (2, 34), (2, 37), … , (2, 101) , (3, 486), …

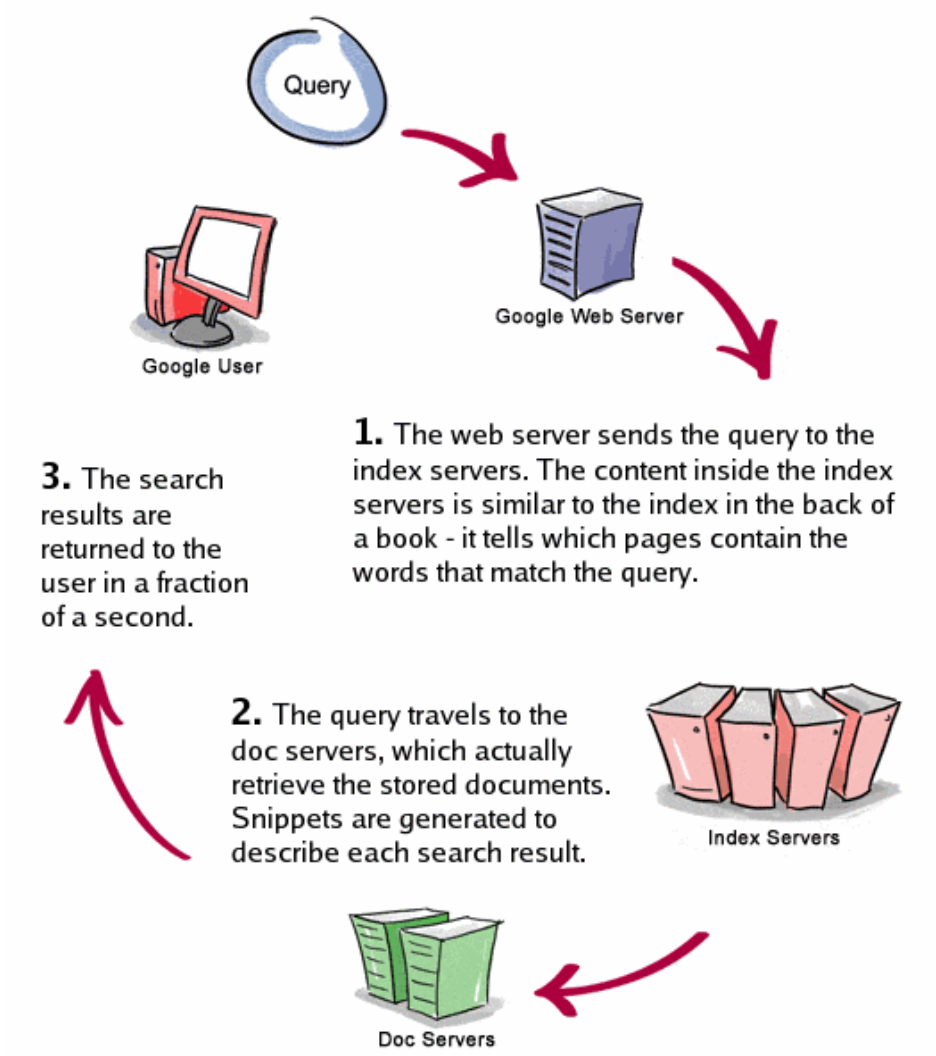
Disk sortering (MapReduce)

(22, 1), (77, 1), . . . , (198, 1) , (1, 2), (22, 2), . . . , (345, 2) , (67, 3), . . .

Inverteret fil

Søgning & Ranking

"Life of a Google Query"



www.google.com/corporate/tech.html

Søgning og Ranking

Søgning: unf AND aarhus

1. Slå **unf** og **aarhus** op i leksikon.
   Giver adresse på disk hvor deres lister starter.
2. Scan disse lister og "flet" dem (returnér DocID'er som er med i begge lister).

unf: 12, 15, 117, 155, 256, ...
aarhus: 5, 27, 117, 119, 256, ...

3. Udregn rank af fundne DocID'er. Hent de 10 højst rank'ede i dokumentsamling og returnér URL samt kontekst fra dokument til bruger.

OR og NOT kan laves tilsvarende. Hvis lister har ord-positioner kan frase-søgninger ("**unf aarhus**") og proximity-søgningner ("**unf**" tæt på "**aarhus**") også laves.

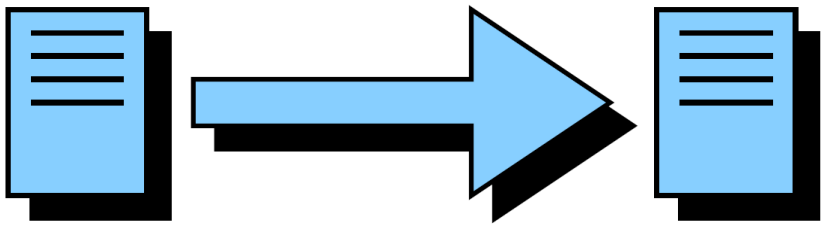
Klassisk Tekstbaseret Ranking

- Vægt forekomsten af et ord med fx
  - Antal forekomster i dokumentet
  - Ordets typografi (fed skrift, overskrift, …)
  - Forekomst i META-tags
  - Forekomst i tekst ved links som peger på siden
- Forbedring, men ikke nok på Internettet (rankning af fx 100.000 relevante dokumenter)

Let at spamme (fyld siden med søge-ord)

Linkbaseret Ranking

- Idé 1: Link til en side ≈ anbefaling af den
- Idé 2: Anbefalinger fra vigtige sider skal vægte mere



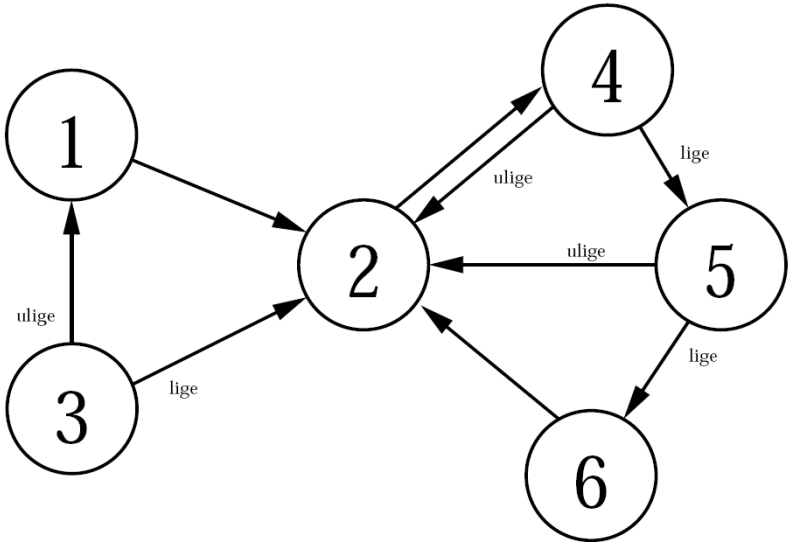
Google PageRank™ ≈ Websurfer

PageRank beregning kan opfattes som en websurfer som (i uendelig lang tid) i hver skridt

- med 85% sandsynlighed vælger at følge et tilfældigt link fra nuværende side,
- med 15% sandsynlighed vælger at gå til en tilfældig side i hele internettet.

PageRank for en side *x* er lig den procentdel af hans besøg som er til side *x*

RandomSurfer



Metode RandomSurfer

Start på knude 1

Gentag mange gange:

    Kast en terning:

    Hvis den viser 1-5:

        Vælg en tilfældig pil ud fra knuden

        ved at kaste en terning hvis 2 udkanter

    Hvis den viser 6:

        Kast terningen igen og spring hen til den knude

        som terningen viser

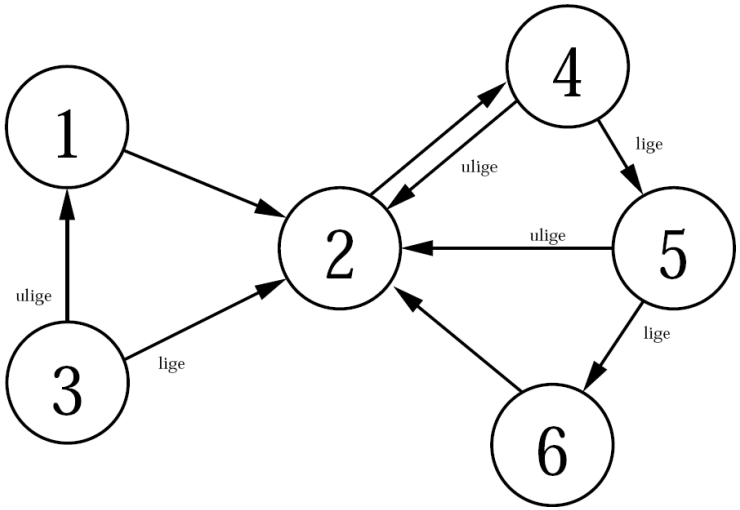
Beregning af Sandsynligheder

Sandsynligheden for at stå i knude *i* efter *s* skridt:

$$p_i^{(s)} = \frac{5}{6} \sum_{j:\, j \to i} p_j^{(s-1)} \cdot \frac{1}{\text{udgrad}(j)} + \frac{1}{6} \cdot \frac{1}{6}$$

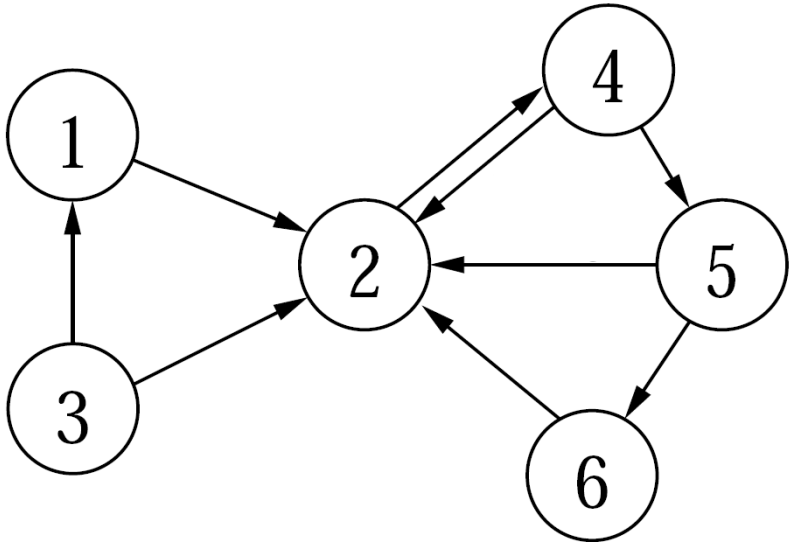
$$p_1^{(0)} = 1.0 \qquad p_2^{(0)} = \cdots = p_6^{(0)} = 0.0$$

Simpel Webgraf — Sandsynlighedsfordeling



Skridt	1	2	3	4	5	6
0	1.000	0.000	0.000	0.000	0.000	0.000
1	0.028	0.861	0.028	0.028	0.028	0.028
2	0.039	0.109	0.028	0.745	0.039	0.039
3	0.039	0.432	0.028	0.118	0.338	0.044
4	0.039	0.299	0.028	0.388	0.077	0.169
5	0.039	0.406	0.028	0.277	0.189	0.060
6	0.039	0.316	0.028	0.366	0.143	0.107
7	0.039	0.373	0.028	0.291	0.180	0.087
8	0.039	0.342	0.028	0.339	0.149	0.103
9	0.039	0.361	0.028	0.313	0.169	0.090
10	0.039	0.348	0.028	0.329	0.158	0.098
11	0.039	0.357	0.028	0.318	0.165	0.094
12	0.039	0.351	0.028	0.325	0.160	0.096
13	0.039	0.355	0.028	0.320	0.163	0.094
14	0.039	0.352	0.028	0.323	0.161	0.096
15	0.039	0.354	0.028	0.321	0.163	0.095
16	0.039	0.353	0.028	0.323	0.162	0.095
17	0.039	0.354	0.028	0.322	0.162	0.095
18	0.039	0.353	0.028	0.322	0.162	0.095
19	0.039	0.353	0.028	0.322	0.162	0.095
20	0.039	0.353	0.028	0.322	0.162	0.095
21	0.039	0.353	0.028	0.322	0.162	0.095
22	0.039	0.353	0.028	0.322	0.162	0.095
23	0.039	0.353	0.028	0.322	0.162	0.095
24	0.039	0.353	0.028	0.322	0.162	0.095
25	0.039	0.353	0.028	0.322	0.162	0.095

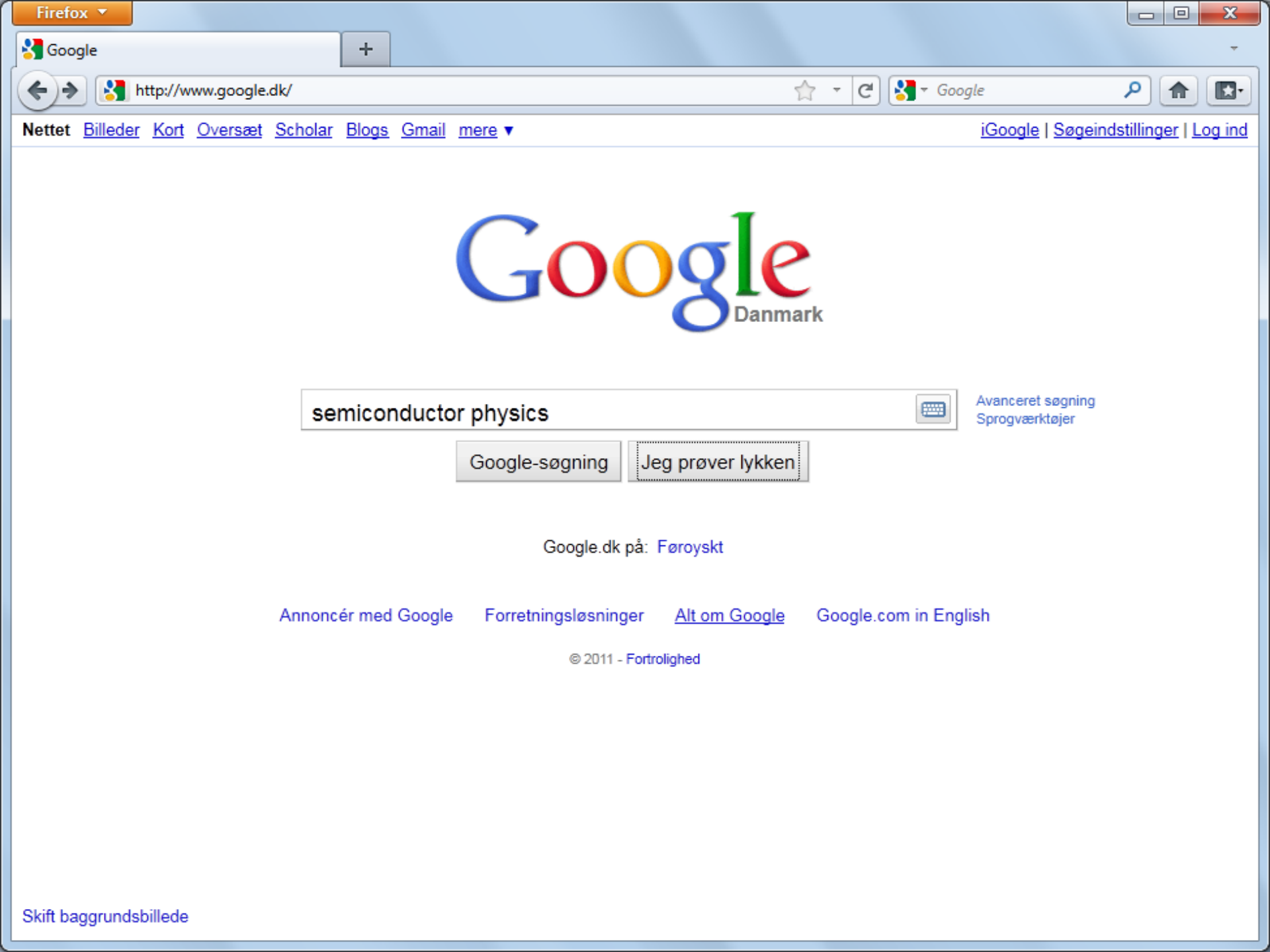
PageRank vektoren = en egenvektor



$$A = 0.85 \cdot \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0.5 & 0 & 0 & 0.5 & 0 \\ 0 & 0.5 & 0 & 0 & 0 & 0.5 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}^T + 0.15 \cdot \frac{1}{6} \cdot \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 0.025 & 0.875 & 0.025 & 0.025 & 0.025 & 0.025 \\ 0.025 & 0.025 & 0.025 & 0.875 & 0.025 & 0.025 \\ 0.45 & 0.45 & 0.025 & 0.025 & 0.025 & 0.025 \\ 0.025 & 0.45 & 0.025 & 0.025 & 0.45 & 0.025 \\ 0.025 & 0.45 & 0.025 & 0.025 & 0.025 & 0.45 \\ 0.025 & 0.875 & 0.025 & 0.025 & 0.025 & 0.025 \end{pmatrix}$$

Find vektoren *p så* ***Ap = p***,

dvs p er en egenvektor for A for egenværdien λ=1 ($Ap = \lambda p$)


Firefox
Google
http://www.google.dk/
Google
Nettet Billeder Kort Oversæt Scholar Blogs Gmail mere ▾
iGoogle | Søgeindstillinger | Log ind
Google
Danmark
semiconductor physics
Avanceret søgning
Sprogværktøjer
Google-søgning
Jeg prøver lykken
Google.dk på: Føroyskt
Annoncér med Google
Forretningsløsninger
Alt om Google
Google.com in English
© 2011 - Fortrolighed
Skift baggrundsbillede

Firefox
Britney Spears' Guide to Semiconductor ...
http://britneyspears.ac/lasers.htm
Google

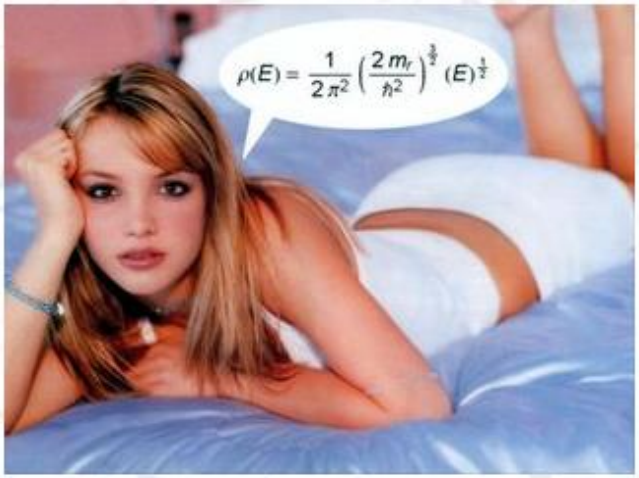
Britney Spears' Guide to Semiconductor Physics

Spectral Optics Offers custom and standard line of laser and optical components. www.spectraloptics.com

I G L Red and green positioning lasers. Industrial quality made in germany. www.igl.de.com

Quattro Titanium Få en glat og behagelig barbering. Modtag en gratis prøve nu! www.efi.dk

Ads by Google



Booble

Booble It

Web britneyspears.ac

Scientific Calculator

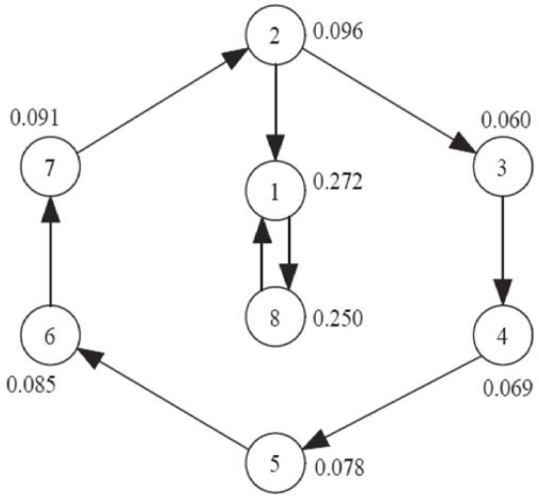
Advertise Here

It is a little known fact, that Ms Spears is an expert in semiconductor physics. Not content with just singing and acting, in the following pages, she will guide you in the fundamentals of the vital semiconductor laser components that have made it possible to hear her super music in a digital format.

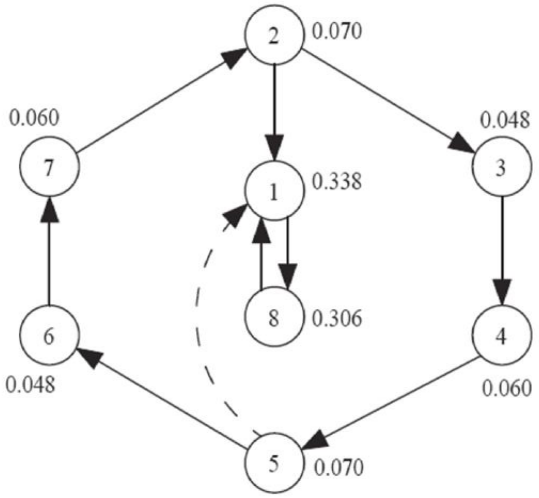
Click here to donate food to the starving people of the world.

the hunger site

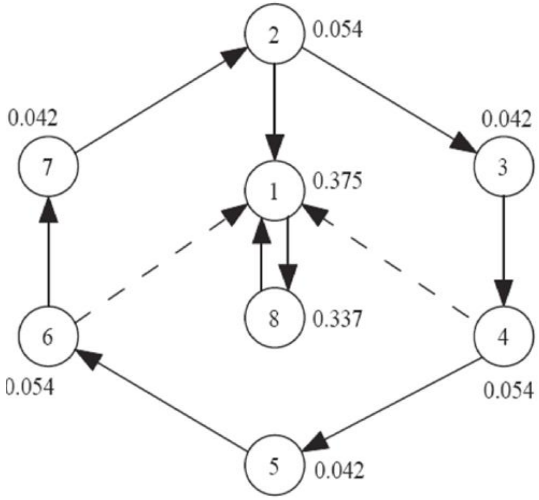
Søgemaskine Optimering



(a) The original graph.



(b) One optimal new link.



(c) Two optimal new links.

Martin Olsen, ph.d. afhandling, august 2009

Søgemaskine Optimering (SEO)

- Mål: Optimer websider til at ligge højt på søgemaskiner
- Metoder: Lav nye websider der peger på en side, bytte links, købe links, lave blog indlæg, meta-tags, ...
- SEO: Milliard industri
- Søgemaskiner: Blacklisting, fjernelse af dubletter, ...

SAS-hoteller sortlistet efter Google-fusk

Verdens mest populære søgemaskine, Google, har boykottet SAS-koncernens nordiske hoteller og konferencecentre, efter de har brugt skjulte websider til at opnå en god placering i søgeresultaterne. Metoden er udviklet af danske Netpointers, som risikerer en bombe under sit forretningsgrundlag.

www.computerworld.dk, 9. november 2004