구글 검색 기술과 페이지랭킹

참고 자료 – http://en.wikipedia.org/wiki/PageRank, 그리고

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

The PageRank Citation Ranking: Bringing Order to the Web

January 29, 1998

Abstract

The importance of a Web page is an inherently subjective matter, which depends on the readers interests, knowledge and attitudes. But there is still much that can be said objectively about the relative importance of Web pages. This paper describes PageRank, a method for rating Web pages objectively and mechanically, effectively measuring the human interest and attention devoted to them.

We compare PageRank to an idealized random Web surfer. We show how to efficiently compute PageRank for large numbers of pages. And, we show how to apply PageRank to search and to user navigation.

1 Introduction and Motivation

The World Wide Web creates many new challenges for information retrieval. It is very large and heterogeneous. Current estimates are that there are over 150 million web pages with a doubling life of less than one year. More importantly, the web pages are extremely diverse, ranging from "What is Joe having for lunch today?" to journals about information retrieval. In addition to these major challenges, search engines on the Web must also contend with inexperienced users and pages engineered to manipulate search engine ranking functions.

However, unlike "flat" document collections, the World Wide Web is hypertext and provides considerable auxiliary information on top of the text of the web pages, such as link structure and link text. In this paper, we take advantage of the link structure of the Web to produce a global "importance" ranking of every web page. This ranking, called PageRank, helps search engines and users quickly make sense of the vast heterogeneity of the World Wide Web.

Searching the Web

Arvind Arasu Junghoo Cho Hector Garcia-Molina Andreas Paepcke Sriram Raghavan
Computer Science Department, Stanford University
{arvinda,cho,hector,paepcke,rsram}@cs.stanford.edu

Abstract

We offer an overview of current Web search engine design. After introducing a generic search engine architecture, we examine each engine component in turn. We cover crawling, local Web page storage, indexing, and the use of link analysis for boosting search performance. The most common design and implementation techniques for each of these components are presented. We draw for this presentation from the literature, and from our own experimental search engine testbed. Emphasis is on introducing the fundamental concepts, and the results of several performance analyses we conducted to compare different designs.

Keywords: Search engine, crawling, indexing, link analysis, PageRank, HITS, hubs, authorities, information retrieval.

"PageRank Uncovered"

Formerly PageRank Explained

Written and theorised by Chris Ridings and Mike Shishigin

Edited by Jill Whalen

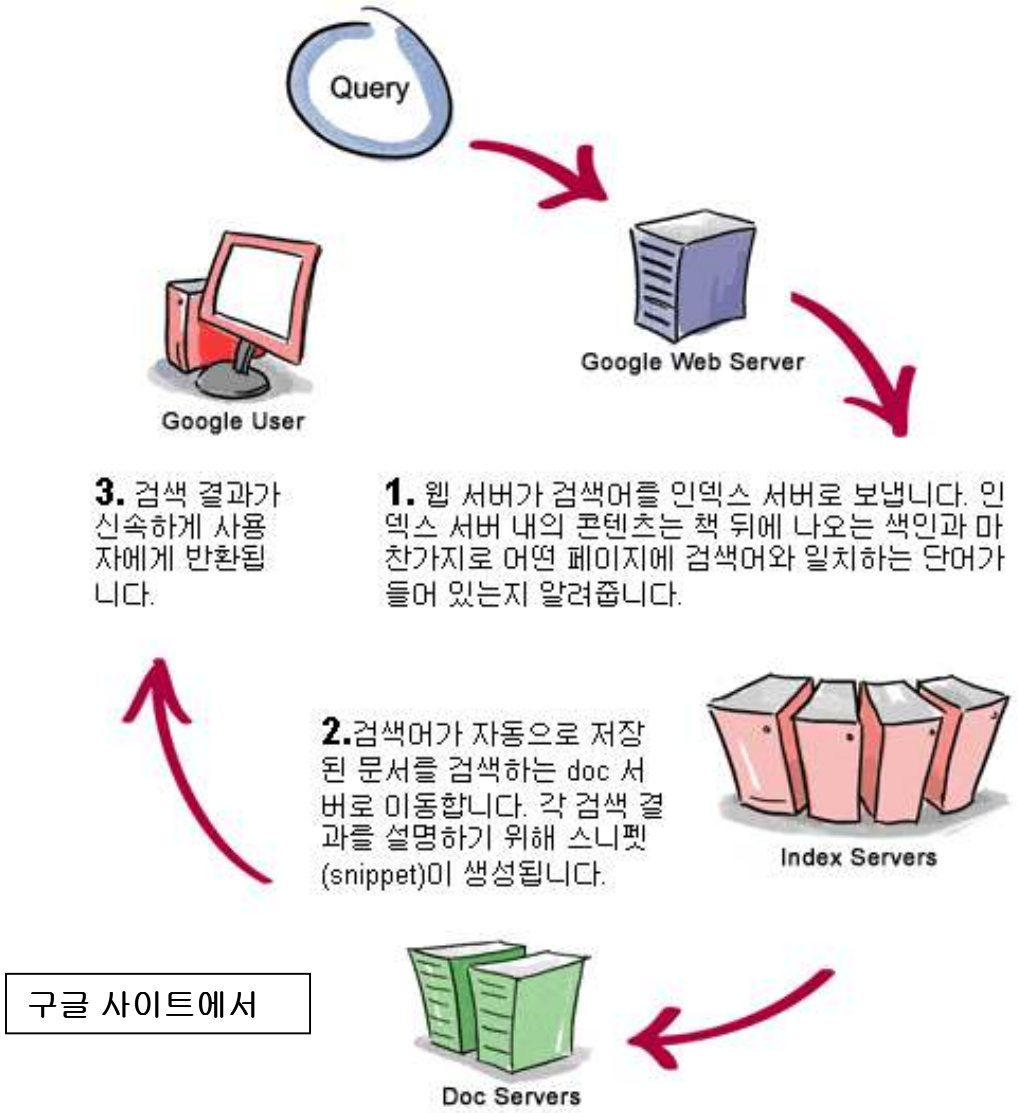
Technical Editing by Yuri Baranov

Profiles:

Chris Ridings is the original author of PageRank Explained. A former programmer of custom search engines, search engine expert, creator of the search engine optimization support forums (http://www.supportforums.org) and author of marketing software tools (http://www.ianbar.com/). Chris uses his programming experience and knowledge of algorithms to get an insight into how search engines work "under the cover."

Mike Shishigin, PhD, is a brilliant search engine expert, whose philosophy is to always back up all his words with hard facts derived from deep knowledge of applied mathematics. As Director of Technology at Radiocom Ltd. Mike uses methods of sampling analysis to get into the nature of things, providing users of WebSite CEO with an outstanding Web site promotion suite (http://www.websiteceo.com); thus providing them with the best possible results in the search engines. Mike controls all R&D and search engine studies within WebSite CEO. His latest "hobby" is the PageRank technology because it is similar to graph theory, his favourite university course. His strongest dislike is search engine spam, which he considers the deadly sin of Internet age. Mike mercilessly fights it by enlightening minds of WebSite CEO users and making regular raids into spammers' camps.

인터넷 검색 101



1. 웹 Crawling - 링크를 따라 가면서 웹페이지 방문
2. 색인 – 페이지내의 모든 단어들을 파싱해서 각 단어별 색인링크에 넣음
3. 검색 단어별 질의 – 검색 단어별로 색인링크를 따라가면서 (모든) 단어가 들어있는 페이지 추출
4. 페이지 순위 매기기 – 가장 적절한 페이지 순서별로 정렬
5. 디스플레이 – 사용자가 이해하기 쉽게 검색결과 보여줌

역사는 밤에 이루어진다 – history is made at night

구글 검색엔진 특징

검색결과 품질을 높이는 2가지 주요 특징 :

- 웹의 링크구조를 이용 (PageRank)
- 링크에 딸린 텍스트 (Anchor text)를 이용해 페이지를 더 정확히 가져옴

이외 다른 특징으로는 :

- 도큐먼트(웹페이지)의 어디에서, 즉 URL 이냐 title이냐 또는 본문이냐, 검색어가 발견되었는지 활용
- 몇 개의 검색어가 있을 때 이들이 페이지내에 서로 얼마나 가까운 곳에서 발견되었는지 활용
- 폰트 크기 등 검색어가 어떤 형태로 웹페이지에서 보이는 지
- 원본 hmtl 페이지를 자체내에 보관, 관리

직관적으로 본 PageRanking – bring order to the Web?

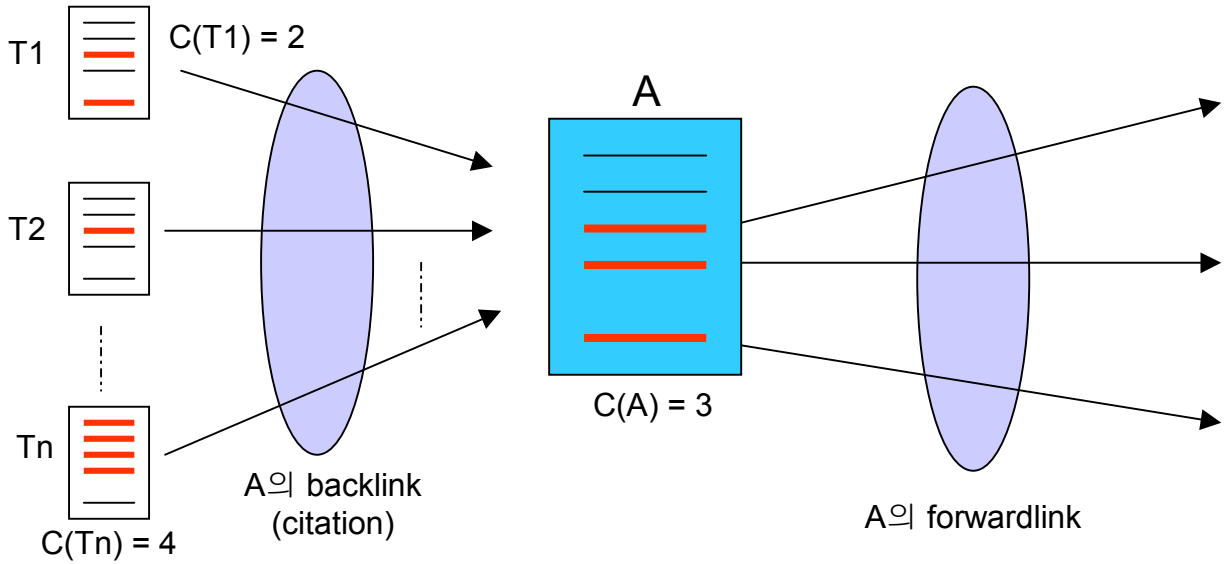
사람사는 세상에서 내가 괜찮은 사람이라 말할 수 있으려면 :

- 나를 보고 괜찮은 친구라 말해주는 다른 사람들이 많고
- 나를 괜찮은 친구라 말해주는 사람 자신이 괜찮은 인물이면 좋고
- 나를 괜찮은 친구라 말해주는 인물이 (사람 평가)가 헤퍼 아무에게나 괜찮은 놈이라 말하지 않을 수록 좋다

웹서핑을 하는 인간이 어떤 웹페이지 방문할 확률이 **PageRanking**에 따른다면 이는 :

- 그 페이지의 **PageRanking**이 높을 수록 웹서퍼가 방문할 가능성이 높고
- 가끔씩은 서퍼가 현 페이지의 링크를 따라가다가 임의의 다른 페이지로 점프한다고 해도,
- 웹전체로 볼 때 결국은 웹의 모든 각각의 웹페이지를 방문할 확률이 고정된 **PageRanking** 값으로 수렴된다.

PageRank for Techies



- 페이지 **A**를 가르키는 다른 페이지 **T1, T2, …Tn** 이 있다고 하자
- **A**의 **PageRank**를 **PR(A)** 라 하고, **Tn**의 **PageRank**는 **PR(Tn)** 이다
- **C(A)** 는 페이지 **A** 내에 있는 링크 수, 즉 **A**의 **forwadlink**의 합이다
- **d**는 **damping** 값 (일종의 튜닝값) 으로 **0 < d < 1** 이나 통상 **0.85** 로 이는 얼마나 다른 페이지로 점프할 경향이 많게 할 지를 나타낸다
- 이 경우, **PR(A) = (1-d) + d•[PR(T1)/C(T1) + …. + PR (Tn)/C(Tn)]**

PageRank 음미

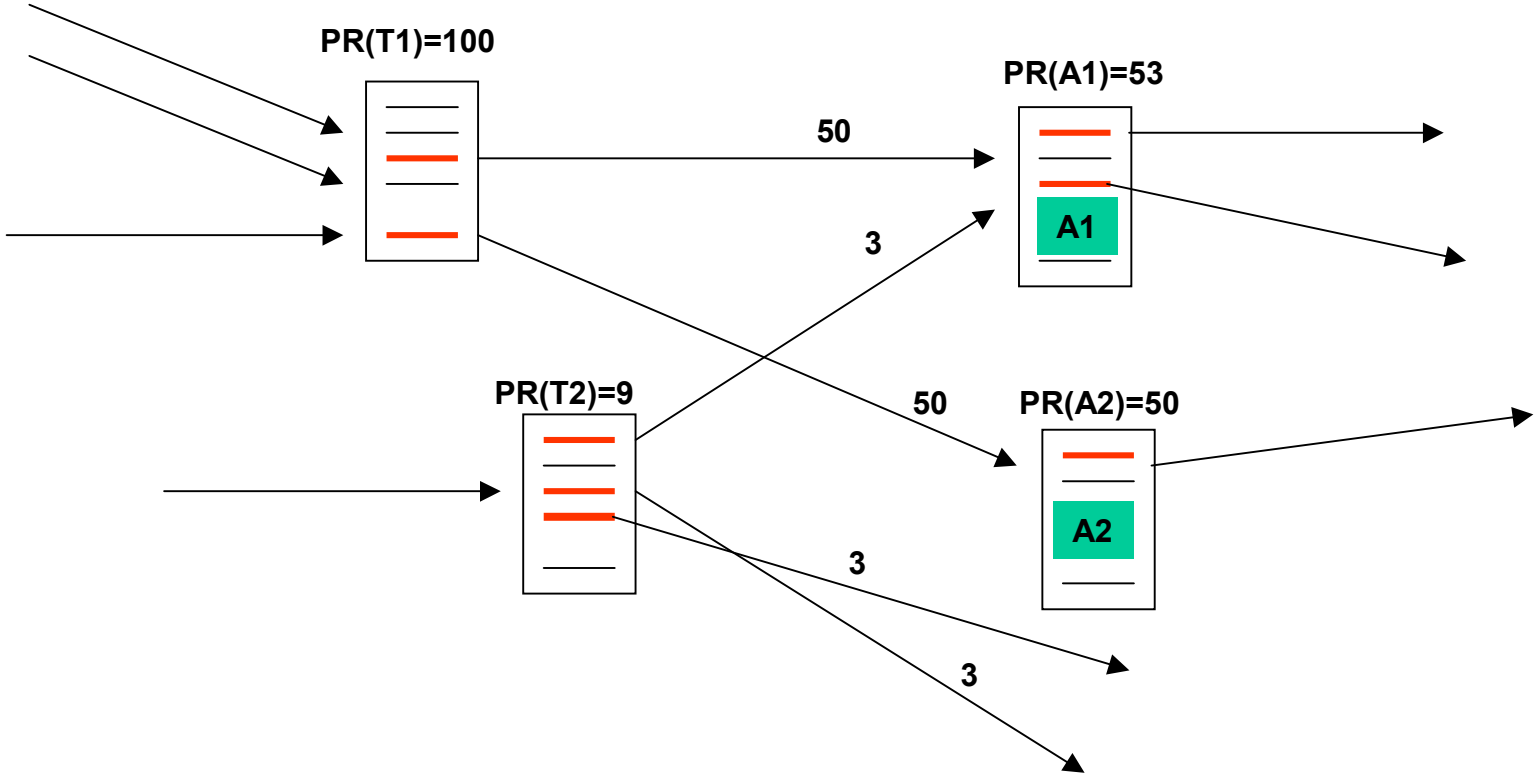
$$PR(A) = (1-d) + d \cdot [PR(T1)/C(T1) + \ldots + PR(Tn)/C(Tn)]$$

= PA, (Peer Assessment) 라 하자

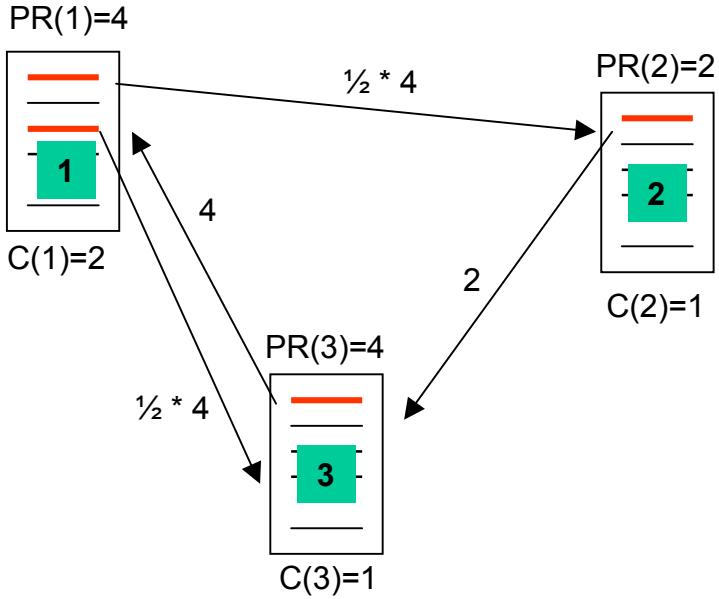
- 쉽다. 크다
- 어떤 페이지의 **PageRank**는 **d** 값에 따라 **PA** 즉 외부평가의 기여도가 바뀐다
- **d** ≠ **0** 이라면, **PA**가 커질 수록 현재 페이지의 **PageRank**가 높아진다
- 현 페이지를 가르키는 **PR(T1)… PR(Tn)**에서 **n**이 많아질 수록, 그리고 각각의 **PR(T1)… PR(Tn)**의 값이 클수록 **PA**가 커진다
- **d** → **0** 일수록 **PA**의 영향력은 줄어들어 이전의 **PR** 값이 그대로 유지되려는 경향이 커진다. 즉 **PageRank**의 변화/응답성이 약해진다. 웹서퍼 관점에서 보면 특별히 중요한 페이지도 없고, 그렇지 않은 페이지도 없다고 느낀다.
- **d** → **1** 이어서 극단적으로 **d=1** 이라면 **PR**은 전적으로 **PA**에 따른다. 즉, **PR**을 업데이트하는 그 당시의 외부의 평가에 **100%** 좌우된다. 기억을 활용하지 않는다. 웹서퍼로 말하자면 가볍게 항상 전혀 관련없는 사이트를 집적거린다
- **PR** 계산이 **recursive** 하다. 수렴이 되나? 실제 적용하면 얼마나 걸리지?

PageRank 단순 예

d = 1 시



PageRank 수렴



$$PR(1) = 0 \cdot PR(1) + 0 \cdot PR(2) + 1 \cdot PR(3)$$
$$PR(2) = 1/2 \cdot PR(1) + 0 \cdot PR(2) + 0 \cdot PR(3)$$
$$PR(3) = 1/2 \cdot PR(1) + 1 \cdot PR(2) + 0 \cdot PR(3)$$

$$\begin{bmatrix} PR \end{bmatrix}_{n+1} = \underbrace{\begin{bmatrix} 0 & 0 & 1 \\ ½ & 0 & 0 \\ ½ & 1 & 0 \end{bmatrix}}_{\text{Link 매트릭스}} \begin{bmatrix} PR \end{bmatrix}_{n}$$

따라서, **link** 매트릭스 구하고 이것에 따른 **PR**
즉, **eigenvector** 구하는 문제가 되었다

1. Link 매트릭스를 $\begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix}$ 하게 놓으면,

k 페이지에서 **j** 페이지로 링크가 있으면 $A_{jk} = \frac{1}{C(k)}$

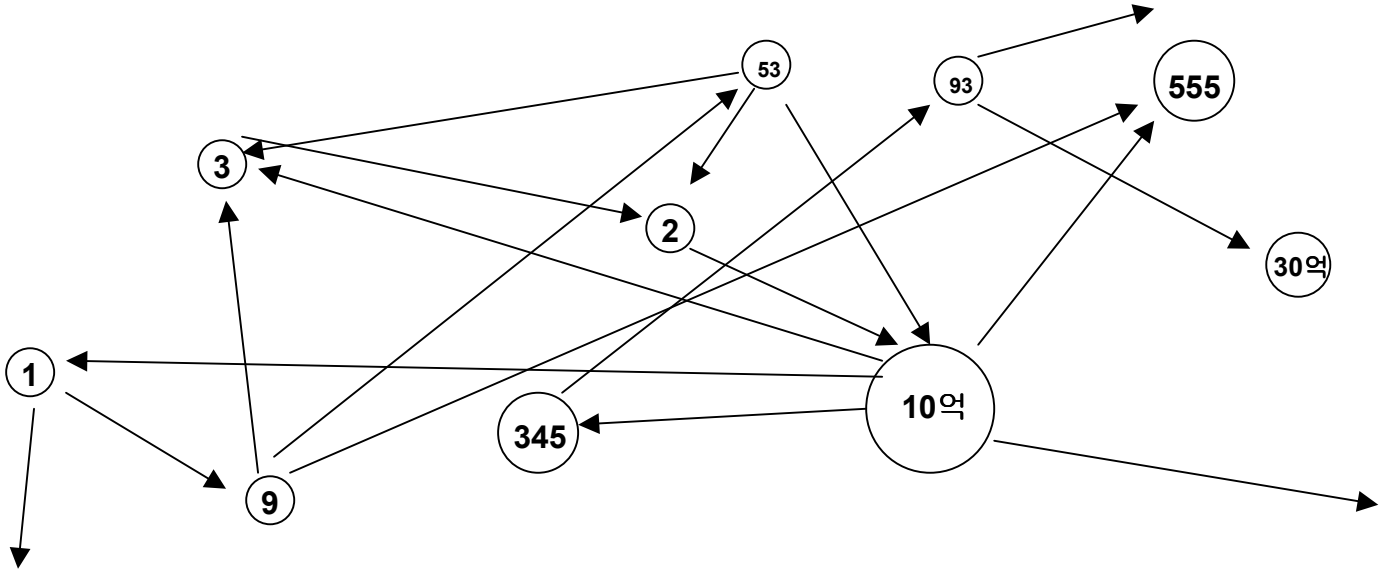
아니면, **0**

2. Link 매트릭스를 $\begin{bmatrix} A_{11} & A_{21} & A_{31} \\ A_{12} & A_{22} & A_{32} \\ A_{13} & A_{23} & A_{33} \end{bmatrix}$ 하게 놓으면,

j 페이지에서 **k** 페이지로 링크가 있으면 $A_{jk} = \frac{1}{C(j)}$

아니면, **0**

단순하게 전세계적으로 생각하면



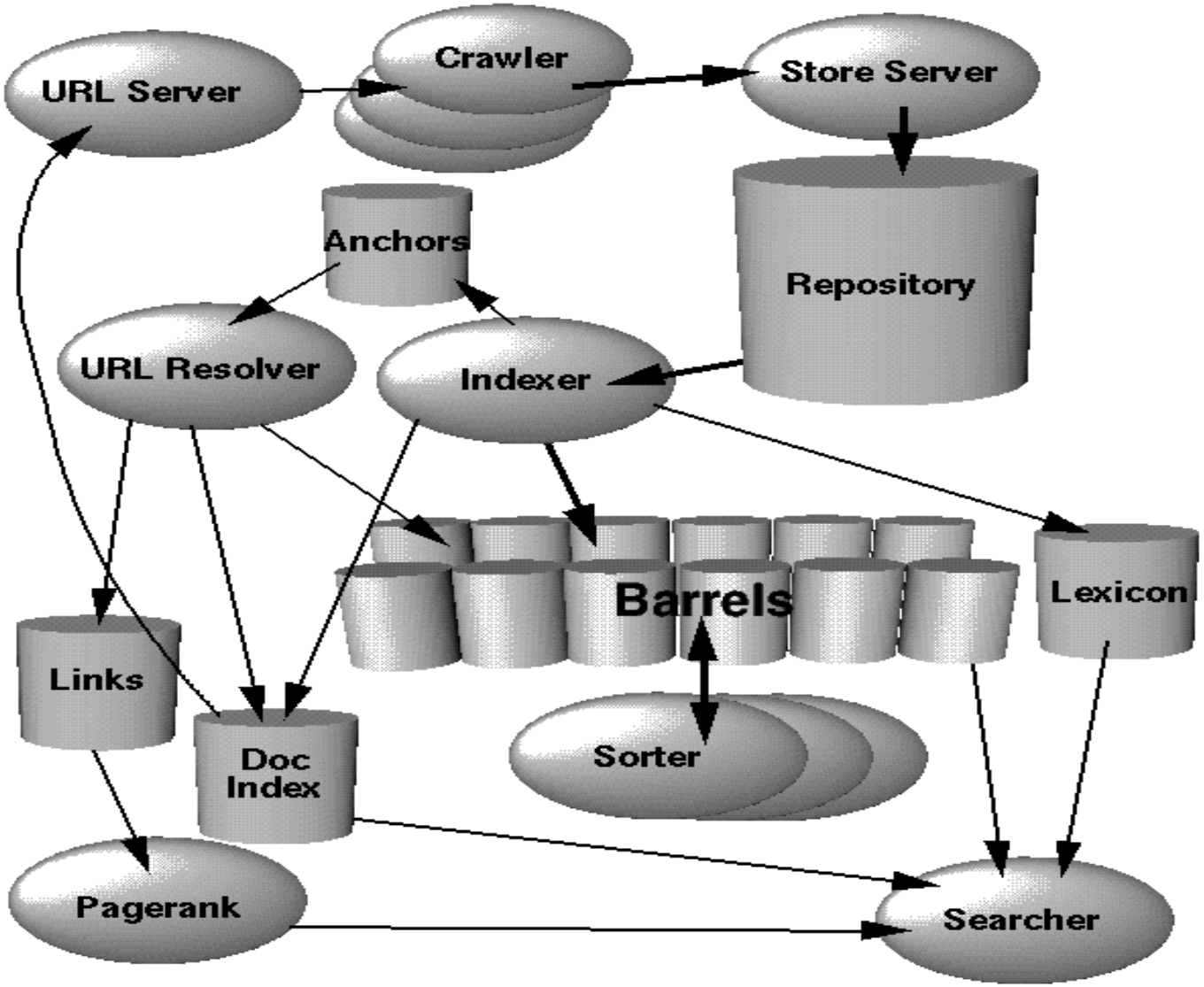
글로벌 웹 그래프

- 30억 (80억) 개의 노드 (인덱싱한 페이지)
- 200억개 이상의 링크

$$\begin{bmatrix} PR(0) \\ PR(1) \\ . \\ . \\ . \\ PR(30억) \end{bmatrix} = \begin{bmatrix} A_{11} & \dots & A_{1,30억} \\ . & & \\ . & & \\ . & & \\ & & A_{30억,30억} \end{bmatrix} \begin{bmatrix} PR(0) \\ PR(1) \\ . \\ . \\ . \\ PR(30억) \end{bmatrix}$$

✓ 링크 매트릭스 **A**는 **coefficient** A_{jk} 가 대부분 **0** 인 **sparse matrix** 이다. 이를 계산에 잘 이용한다

구글 architecture



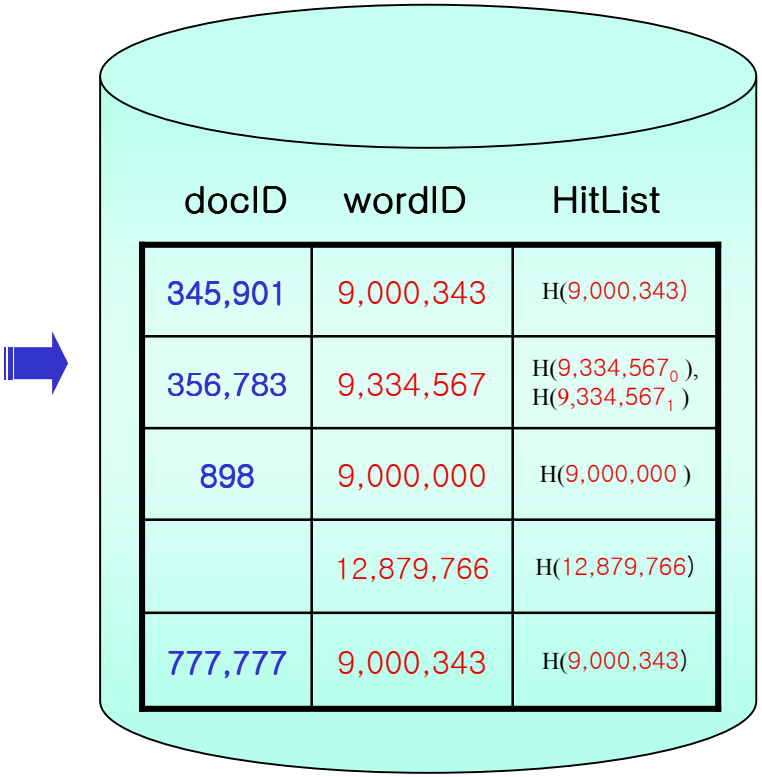
구글 architecture 에서 사용하는 주요 데이터구조

- docID : crawl해서 갖고온 모든 웹페이지에 붙이는 고유 번호 (integer). 웹페이지의 URL을 이용해 생성
- wordID : 사전 (lexicon)에 있는 모든 단어(word)들에 붙이는 고유 번호
- hit : < wordID, 페이지내 단어 위치, 폰트 크기, 대/소문자 구분, descriptor type(anchor ll title ll body ll etc) >. 특정 웹페이지(docID)에서 특정 word(wordID)가 발견되어 그것이 어떤 형태인지를 나타내는 meta 정보를 담고 있는 tuple/array.
- HitList : 특정 document/페이지 내에서 특정 단어가 발견되는 것을 나타내는 hit 들의 리스트. 따라서 HitList를 따라가면 특정 document에서 그 단어가 몇 번 어떻게 나타나는가를 알수 있다. Indexer가 만들어 barrel에 집어 넣는다.
- links : < $docID_{from}$, $docID_{to}$ >. Document간의 링크를 표시
- Doc Index : docID순서에 따라 정렬된 색인으로 docID가 나타내는 웹페이지의 정보 또는 포인터를 담고있다.
- Lexicon : 수천만개의 단어를 담고 있다. 메모리에 위치 (list 또는 Hash Table)
- Barrel (통) : 무슨 통인가 하면 Hit/HitList 를 담는 통인데, 각 통마다 자기가 담을수 있는 word 범위가 있다. 예를 들면, 32번째 통은 "m (wordID=900만)" 에서 "mozart (wordID=13,209,007)" 의 단어를 나타내는 HitList만 담을 수 있는 구조다. 1998년에는 64개의 barrel을 썼다. 내부 데이터구조는?

구글 architecture 에서 사용하는 주요 데이터구조 - 계속

- Forward Index : 웹페이지/document (docID)에 따라 어떤 word 들이 있는 지 나타낸다. 구글에서는 Indexer가 HitList들을 barrel에 넣을 때, word에 따라 맞는 barrel에 집어 넣으므로 이 때 이미 barrel간 상당한 정렬이 되어있다고 할 수 있다.

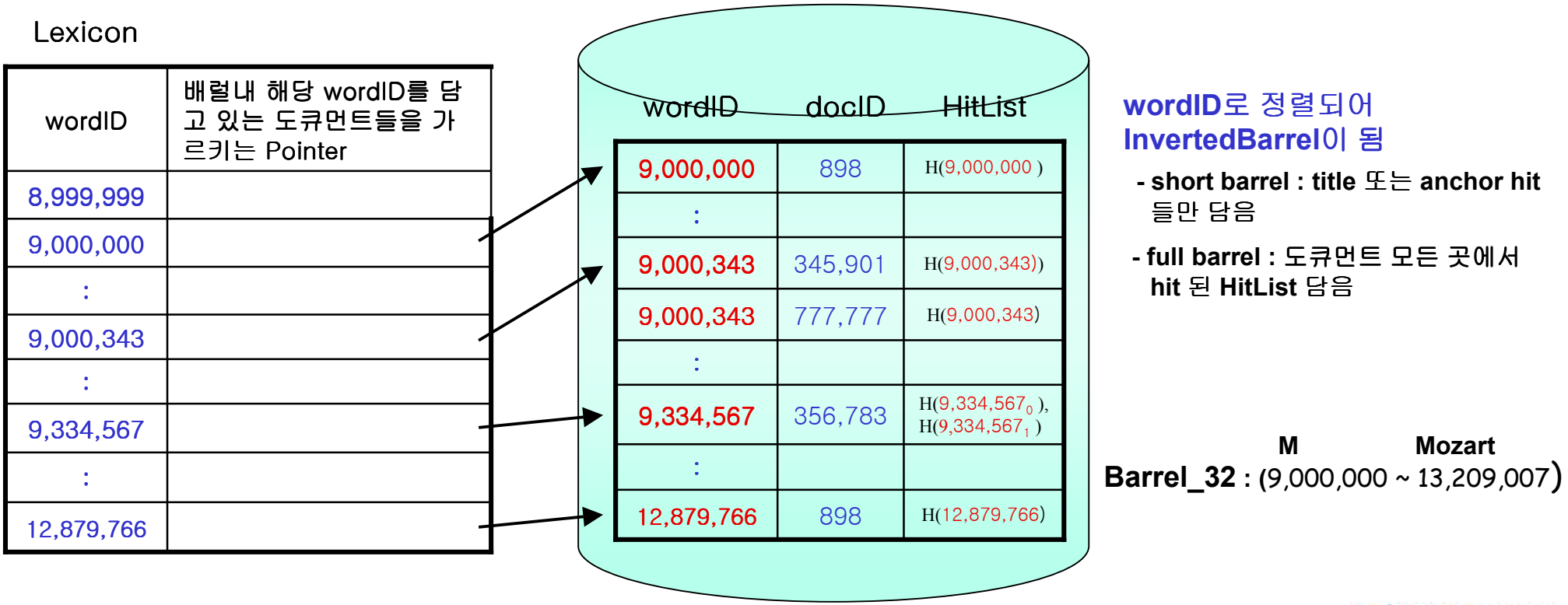
docID	wordID
345,901	345, 876, …. 9,000,343
456,900	23, 78787, 454353, 99
356,783	90,887, 898, $9{,}334{,}567_0$, 234,214, … ,$9{,}334{,}567_1$
898	9,000,000, 9,099 …. 898, 12,879,766, …..
777,777	345,999,878 …….. 9,000,343



Barrel_32 : (M 9,000,000 ~ Mozart 13,209,007)

구글 architecture 에서 사용하는 주요 데이터구조 – 계속

- Inverted Index : wordID를 보면 이 word를 담고 있는 document가 어떤 것들인지 모두 나열해야 한다. Inverted Index는 Forward Index를 wordID를 기준으로 sorting 함으로 생성된다. 이로써 barrel내의 구조는 wordID에 따라 거기에 딸린 document들이 어떤 것들인지 docList 형태로 구성된다. Sorting은 Sorter가 barrel에 들어있는 데이터들에 대해 틈틈이 수행한다. 메모리에 있는 Lexicon은 wordID에 따라 정렬된 < wordID, → > tuple로 되어있는데, → 는 wordID에 따라 정렬된 docList의 첫번째 element를 가르키게 된다.



wordID로 정렬되어 InvertedBarrel이 됨

- **short barrel : title 또는 anchor hit 들만 담음**
- **full barrel : 도큐먼트 모든 곳에서 hit 된 HitList 담음**

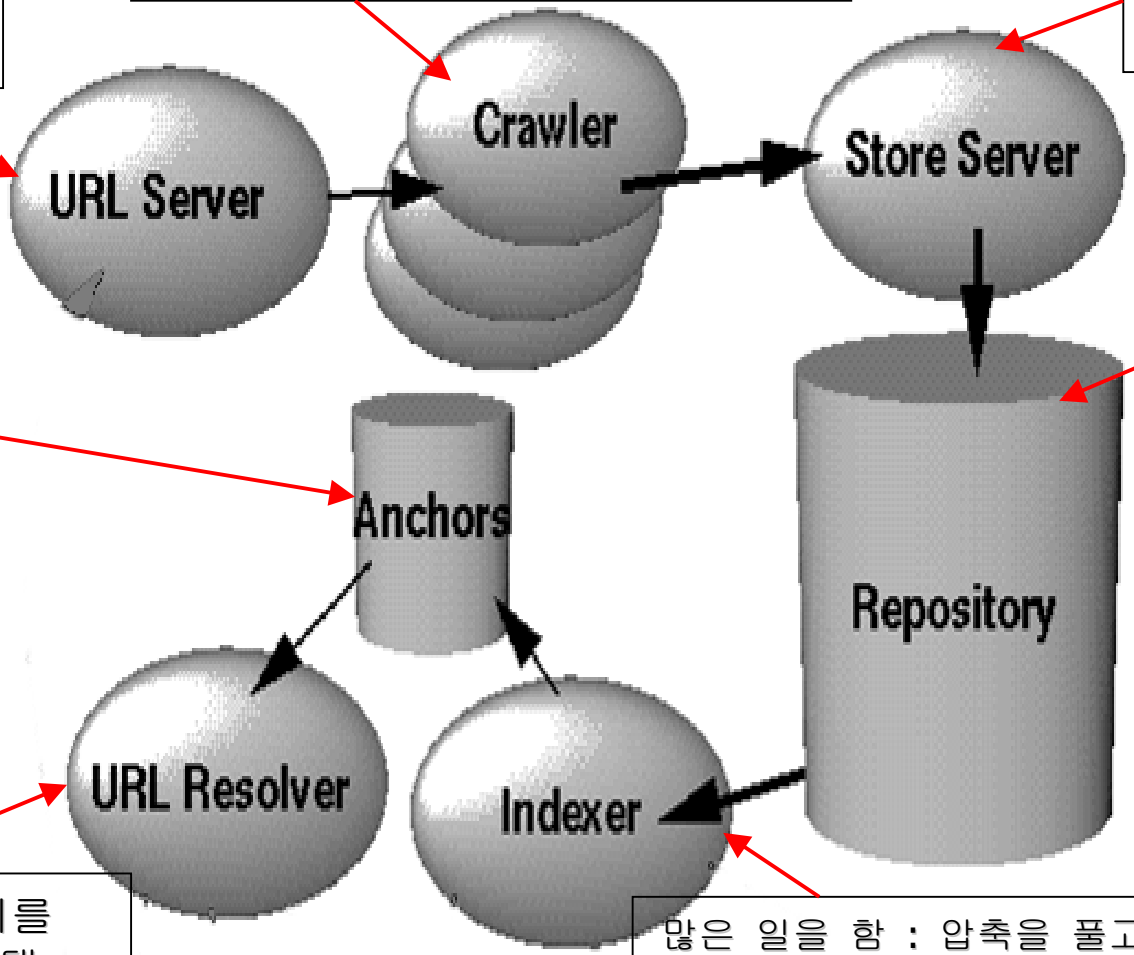
Barrel_32 : (9,000,000 ~ 13,209,007) M Mozart

Crawling, Parsing, and storing

현재 보유하고 있는 URL과 crawl해야 하는 URL 관리

다수의 crawler가 병렬. 각 crawler 가 독자적인 DNS 캐시를 갖고 있으며 약 300개의 커넥션을 동시에 맺을 수 있음

Crawl 해 갖고 온 문서를 압축. 이를 Repository에 넘김



링크들을 관리. 링크에 귀속된 anchor text 저장

모든 웹페이지들의 저장소. Full html 형태로 갖고 있고 각 document들은 docID, 길이, URL로 prefix되어 있음.

Anchor 파일을 읽어 이를 토대로 상대 URL을 절대 URL로 변환. 절대 URL로 docID 생성

많은 일을 함 : 압축을 풀고, document에 대해 파싱. Document에 있는 단어 (Hit)들을 생성. 웹페이지에 있는 모든 링크들을 파싱해 이를 Anchors에 담음.

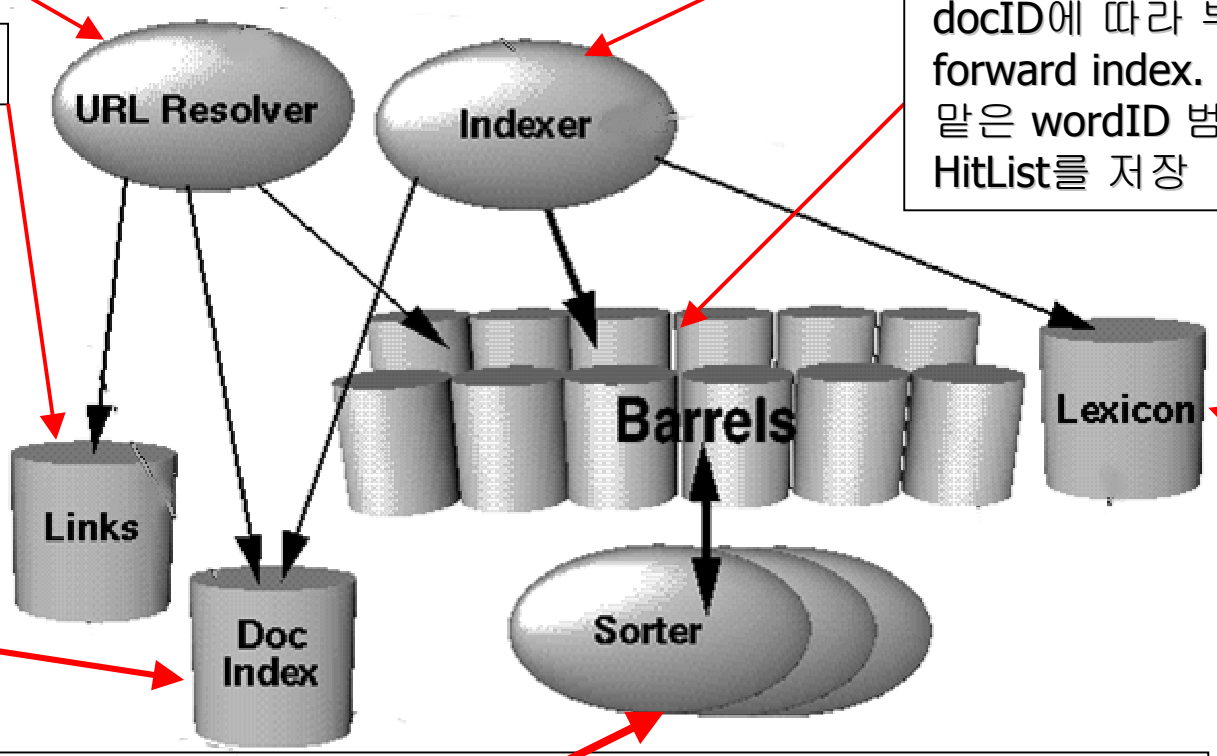
Indexing

생성한 docID를 Doc Index로 보냄. Anchor text를 Barrels에 저장. 링크들을 <docID, docID> 형태로 생성. Anchor text를 anchor text 가 가르키는 docID에 붙여 Barrel로 보냄.

HitList들을 wordID에 따라 맞는 Barrel에 저장

링크들을 담음

docID에 따라 부분적으로 정렬된 forward index. 각 barrel은 자신이 맡은 wordID 범위에 들어온 HitList를 저장

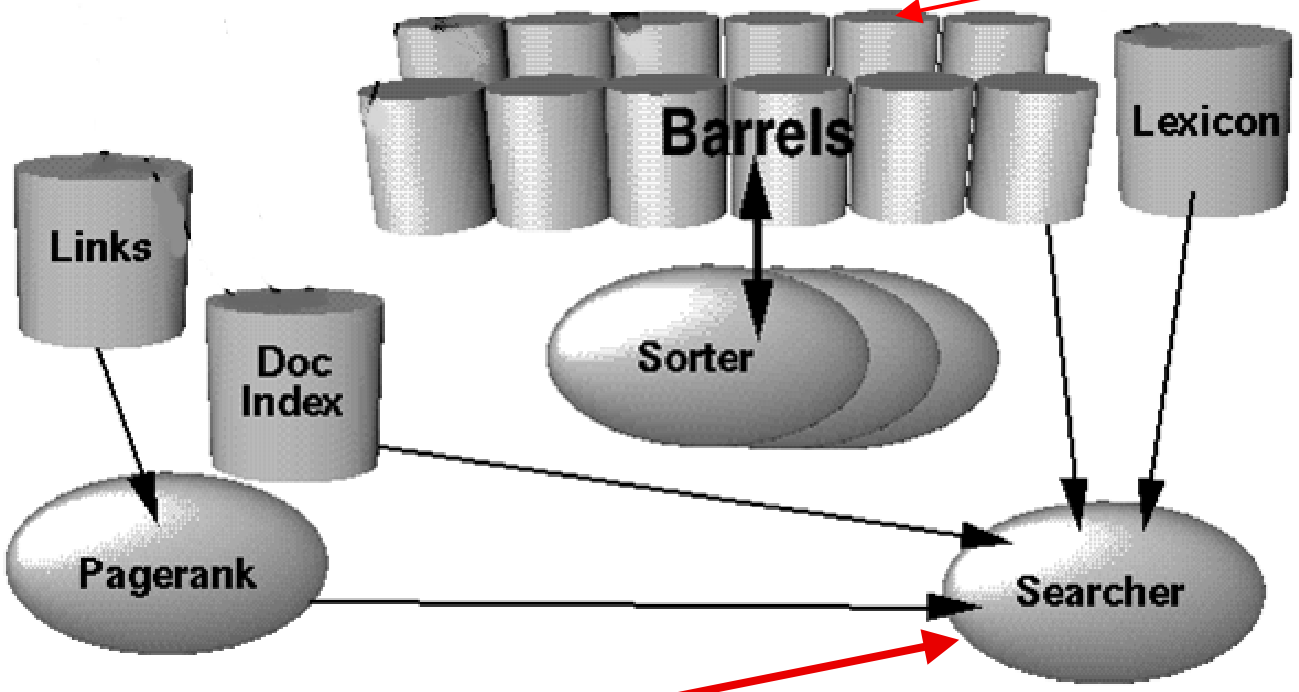


Document의 전반적인 메타정보 관리를 담당. DocID 기준으로 생성된 색인으로 각 엔트리는 Repository내의 해당 Document를 가르키는 포인터, checksum, 통계치, document 상태, URL 등을 나타낸다.

메모리에 위치해 word를 wordID로 매핑. 해당 wordID에 해당되는 Barrel내의 docList를 가르킴

부분정렬된 forward index를 바탕으로, wordID로 정렬해서 inverted index 생성. 이를 바탕으로 wordID가 가르키는 document list로 가서, 해당 word를 담고있는 docID 들과 HitList에 접근한다.

Searching



2가지 Barrel 형태. Short Barrel은 title과 anchor hit 들만을 담은 HitList로 구성. Long Barrel은 그외 모든 장소에서의 HitList

검색이 들어오면, 검색어를 **wordID (Lexicon)**로 바꾸어, 이를 바탕으로 **Barrel (Inverted Index)**로 질의해서 이 검색어들을 갖고 있는 문서들을 검출한다. **DocID**로 **Doc Index**를 활용해 **Repository** 에 있는 웹페이지 본문과 그외 메타정보들에 접근하고, 동시에 **PageRank** 는 이 메타정보들과 **docID**를 활용해 어떤 페이지가 적절한 페이지인가 **PageRank** 계산 (참조). **PageRank**의 결과와 **Repository**에서 갖고 온 데이터로 어떤 웹페이지를 우선 보여줄 가 결정 후, 예쁘게 가공해서 사용자에게 보여 줌.

Google Search Evaluation

1. 질의를 파싱해서
2. 검색어를 wordID로 바꾼 후
3. 각 검색어에 대해 short barrel내의 doclist의 시작으로 가서
4. Doclist들을 스캔하면서 모든 검색어들을 다 담고 있는 도큐먼트(웹페이지)가 있는 가를 (교차하면서) 살펴보고, 있으면
5. 그 페이지의 PageRank를 계산하고
6. 현재 아직 short barrel에 있는 데, 벌써 어떤 doclist의 끝에 와 있다고 하면, 각 단어에 해당하는 full barrel의 시작점으로 간 후 4 번으로 점프
7. 아직 어떤 한 개의 doclist의 끝에도 오지 않았다면 4번으로 점프
8. 검출된 도큐먼트들을 Page Rank에 따라 정렬해서 위부터 k개 return

Multi word search

- 우선 각각의 **word**에 대하여 **Hitlist** 획득

1. 각각의 **hit**는 여러 형태로 생긴다 : **title, anchor, URL,** 큰 폰트, 작은 폰트 등
2. 각각의 **hit type** 은 고유의 **weight (type-weight)**를 갖는다
3. 여러 개의 **hit**가 생겼을 경우 **hit** 간의 거리 **(proximity)**를 계산한다. 구글은 **proximity**를 **10**가지로 나눔. 따라서, 한 문서내에 다수 **hit**가 생겼을 시 **hit** 타입에 따라, 그리고 **hit**간의 **proximity**에 따라 다양한 형태의 점수를 매길 수 있다. 구글은 이를 **type-prox-weight**라 함. 이들을 **vector (1**차원 행렬/배열**)**로 표현한다면

   예) **[TPW[0], TPW[1], …. TPW[n]]**
4. 각 **type**에 따라 **hit** 수가 얼마나 되는 지 **HitList**에서 살펴봄
5. 각 **type**에 따른 **hit** 수들을 세고, 이들 **hit**간의 **Proximity**에 따라서 **weight**를 주어 이를 **count-prox-weight vector** 로 표현

   예) **[CPW[0], CPW[1], … CPW[n]]**
6. **Type-prox-weight** 벡터와 **count-prox-weight** 벡터를 **dot product** 하여 **Information Retrieval (IR)** 점수를 구한다.

   IR 점수 = **TPW ◦ CPW = TPW[0]• CPW[0] + TPW[1]• CPW[1] + ….. + TPW[n]• CPW[n]**
8. **IR** 점수와 **PageRank** 점수를 결합해 최종 순위를 정한다

페이지랭킹과 구글 검색 결과와의 관계는

1. 페이지랭킹은 검색어와 관련이 없다

2. 조금 머리쓰면 페이지랭킹을 올릴 수 있다

3. 듣자하니 페이지랭킹은 IR 값, 즉 직접적 검색어에 따른 검색결과에 곱하기로 작용한다

   즉,

   최종_검색순위 = IR * PageRanking

4. 따라서, 아무리 어떤 페이지가 검색어와 밀접한 내용을 담고 있는 딱 맞는 페이지라 해도, 구글이 그 페이지의 페이지랭킹을 0으로 하면 황이다. 또 아무리 페이지랭킹값이 높아도 IR 값이 최소한을 만족하지 않으면 최종_검색순위에서 좋은 값을 갖기가 어렵다