What is a MapReduce?

Michael Kleber

with many slides shamelessly stolen from Jeff Dean and Yonatan Zunger

Google™

Word Count and friends

Input: (URL, contents)

Map output	Reduce output
(“the”, “1”)	(“the”, “1048576”)
(“the”, “www.bar.org/index.html”)	(“the”, list of URLs)
(hostname, doc term frequencies)	(hostname, site term frequencies)
(“purple”, “cow”)	(“purple”, probability distrib of next word in English)

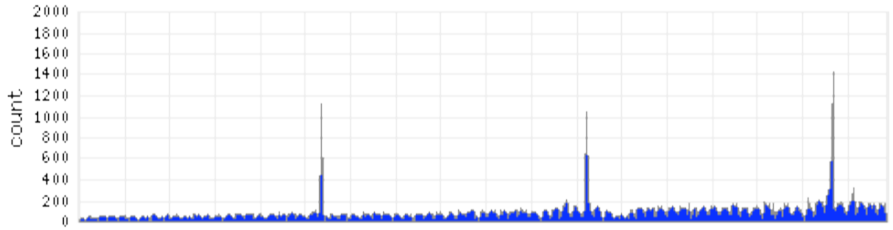
Mappers can take command-line arguments, eg a regular expression.

(“line number”, “matching line”)	(“line number”, “matching line”)

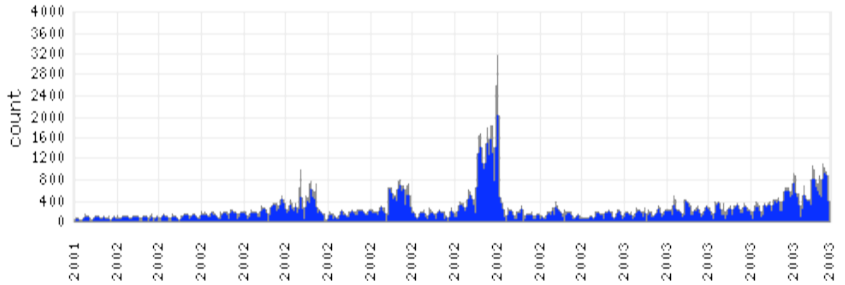
With 1800 machines, MR_Grep scanned 1 terabyte in 100 seconds.

Query Frequency Over Time

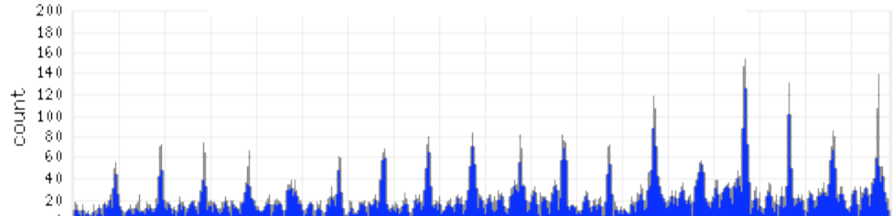
Queries containing “eclipse”



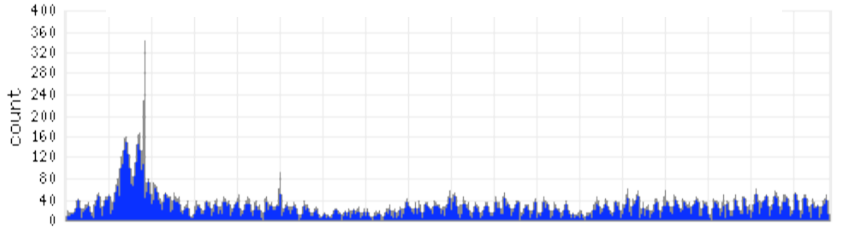
Queries containing “world series”



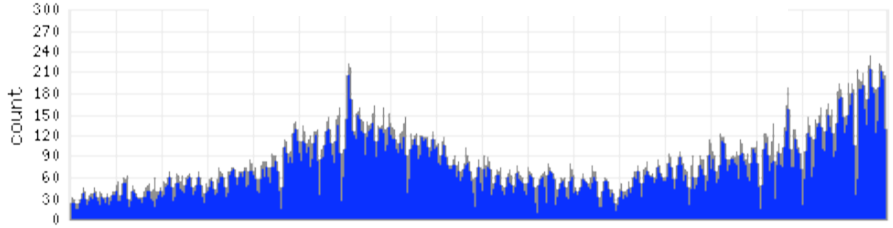
Queries containing “full moon”



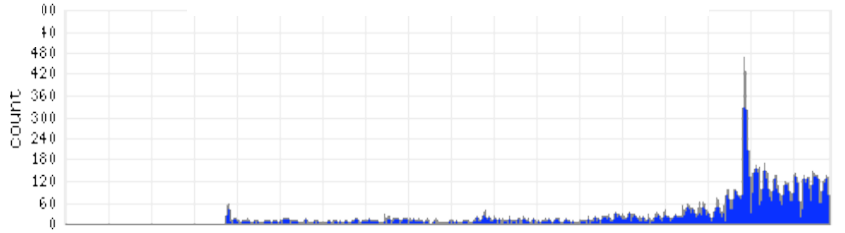
Queries containing “summer olympics”



Queries containing “watermelon”



Queries containing “Opteron”



Sorting

In addition to map and reduce functions, you may specify

Partition: (k', number of reducers) ↦ choice of reducer for k'

Default partition is (hash(k') mod #reducers), for load balancing, but:

- Output file for k' reduction determined by its partition
- Guarantee: each reducer sees the keys in its partition in sorted order (implemented by the invisible shuffle-and-sort stage)

Map: produces (sort key, record)

Partition: send consecutive blocks of sort keys to same reducer,
e.g. by using most significant bits of sort keys

Reduce: identity function

MR_Sort sorted 1 terabyte of 100-byte records in 14 minutes

Structure of the Web

Input is (URL, contents) again

Scan through the document's contents looking for links to other URLs

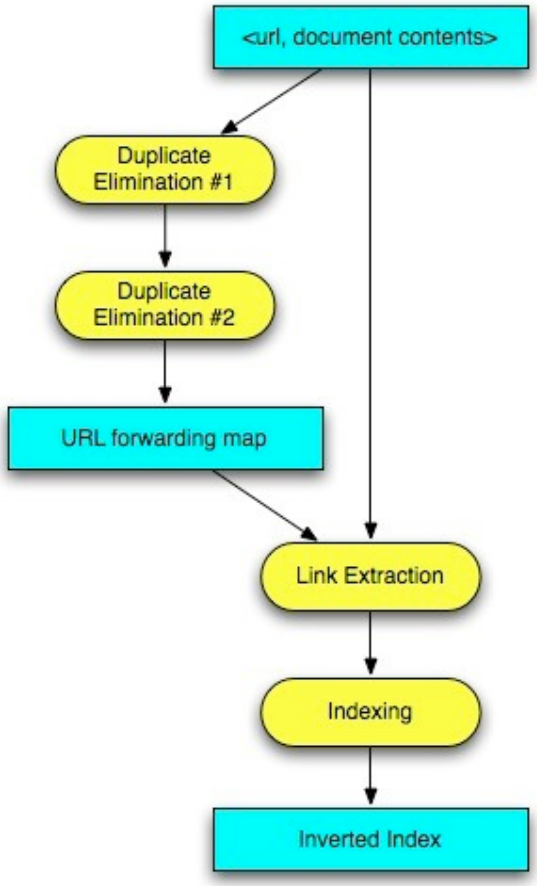
- Map outputs (URL, linked-to URL)
  you get a simple representation of the WWW “link graph”

- Map outputs (linked-to URL, URL)
  you get the reverse link graph, “what web pages link to me?”

- Map outputs (linked-to URL, anchor text)
  you get “how do other web pages characterize me?”

  Google uses this “anchor text propagation” in indexing.

Example: A Simple Indexing Pipeline

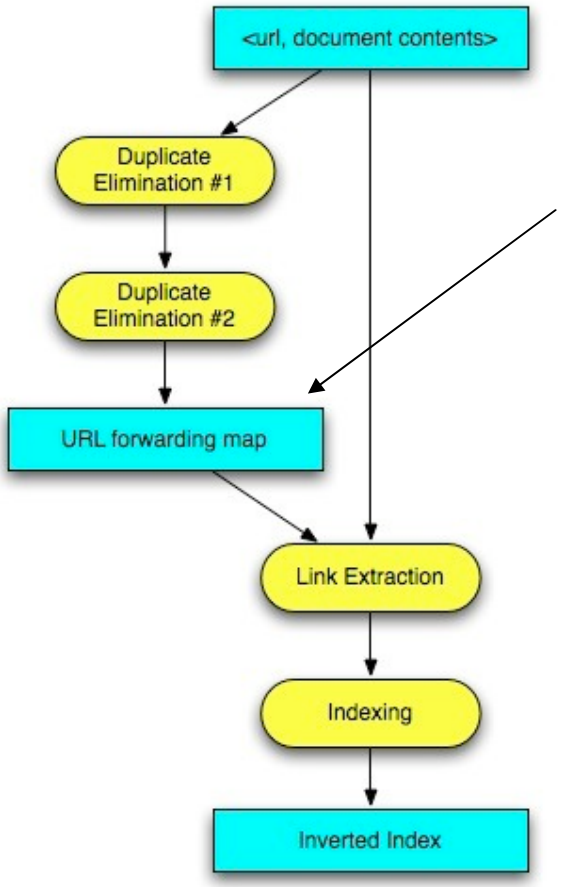


Input: (url, document) pairs

Output: Inverted index including anchor text

Google™

Example: A Simple Indexing Pipeline

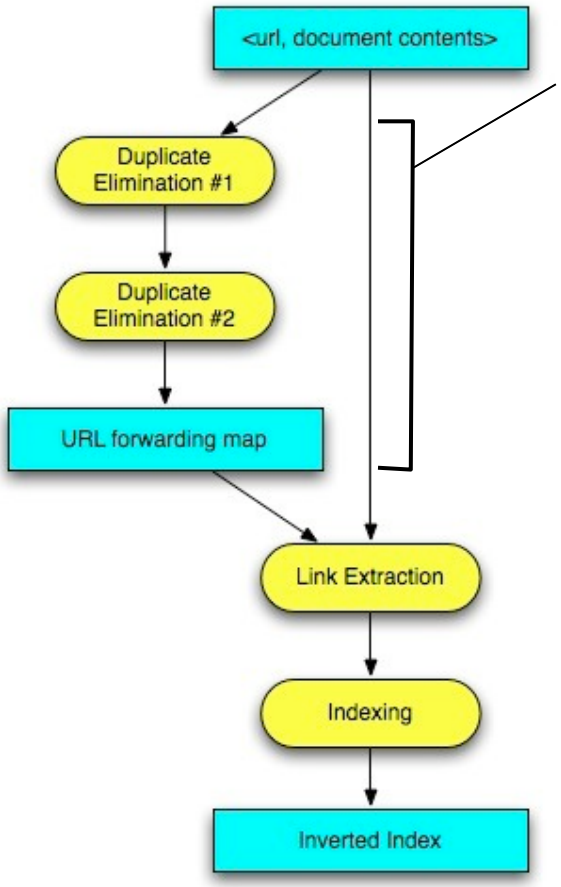


Duplicate Elimination

Want: a sorted table of
(original URL, canonical URL)
for each URL ≠ its canonical URL

Google™

Example: A Simple Indexing Pipeline



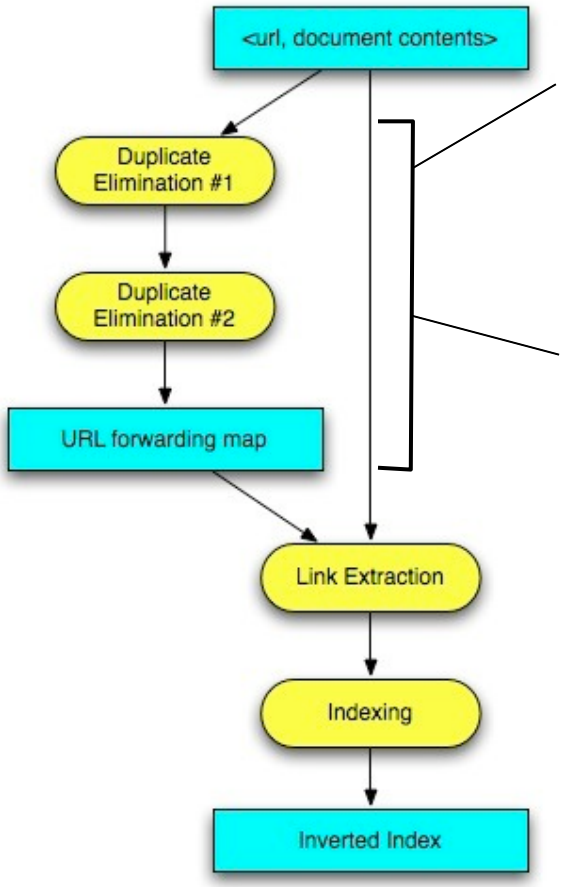
Duplicate Elimination

Map: (url, contents) ↦ <(contentfp, url + aux info)>

Reduce: (contentfp, <url + info>) ↦
<(contentfp, url + "best" url)>

Google™

Example: A Simple Indexing Pipeline



Duplicate Elimination

Map: (url, contents) ↦ <(contentfp, url + aux info)>

Reduce: (contentfp, <url + info>) ↦
<(contentfp, url + "best" url)>

Reorganize that

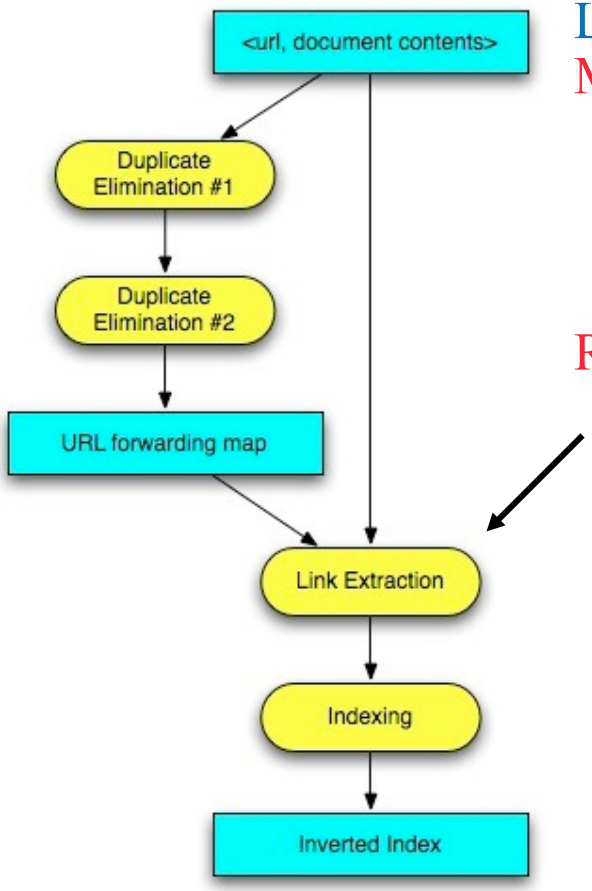
Map: (contentfp, url + canonical url) ↦
(url, canonical url) (or nothing if equal)

Reduce: Identity

Partition: hash(url) % number of output shards

Google™

Example: A Simple Indexing Pipeline



Link Extraction

Map: (url, contents)

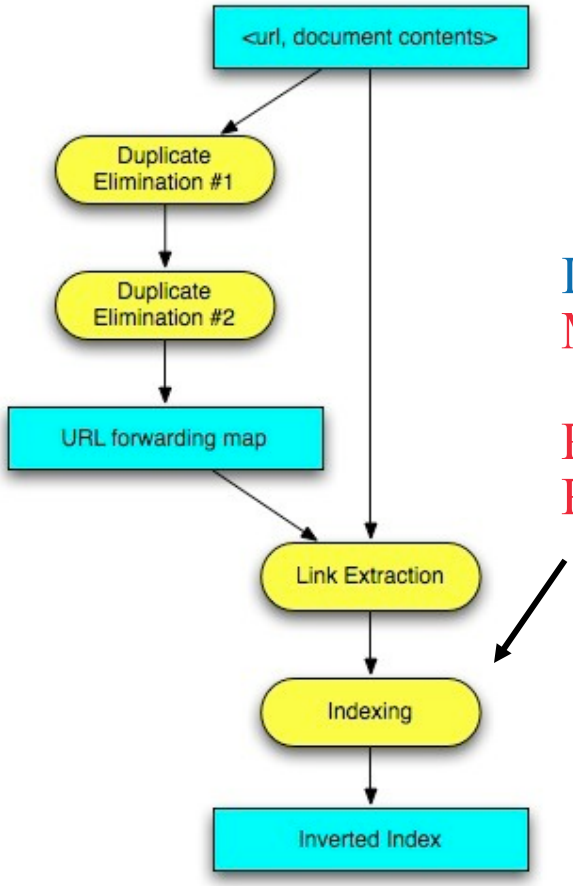
(a) Parse contents, forward each link using map; output (target url, anchor info)

(b) Try to forward doc URL using map; if it doesn't forward, output (url, contents)

Reduce: (forwarded url, <content or anchor>) ↦ (url, contents + anchors)

Google™

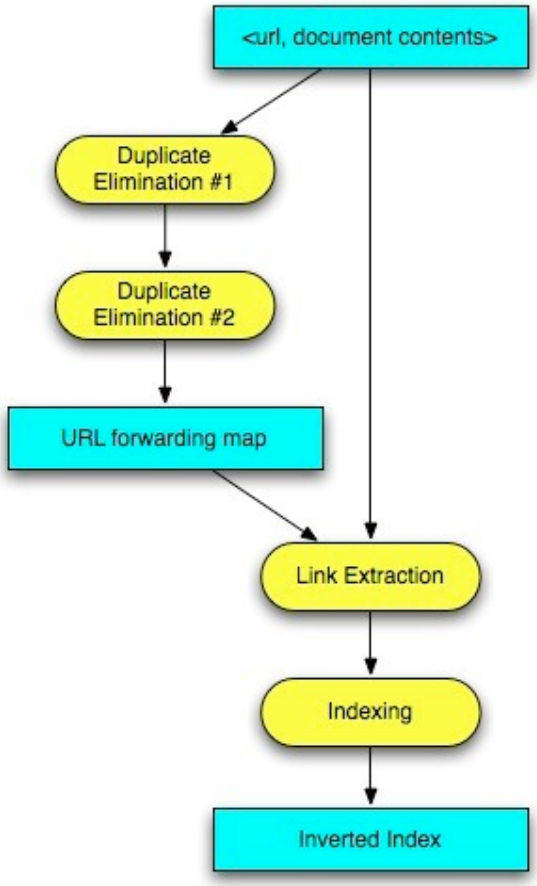
Example: A Simple Indexing Pipeline



Indexing
Map: (url, merged contents) ↦
<(term, occurrence info)>
Partition: hash(url) % number of output shards
Reduce: (term, occurrences in partition of docs) ↦
(term, compressed posting list)

Google™

Example: A Simple Indexing Pipeline

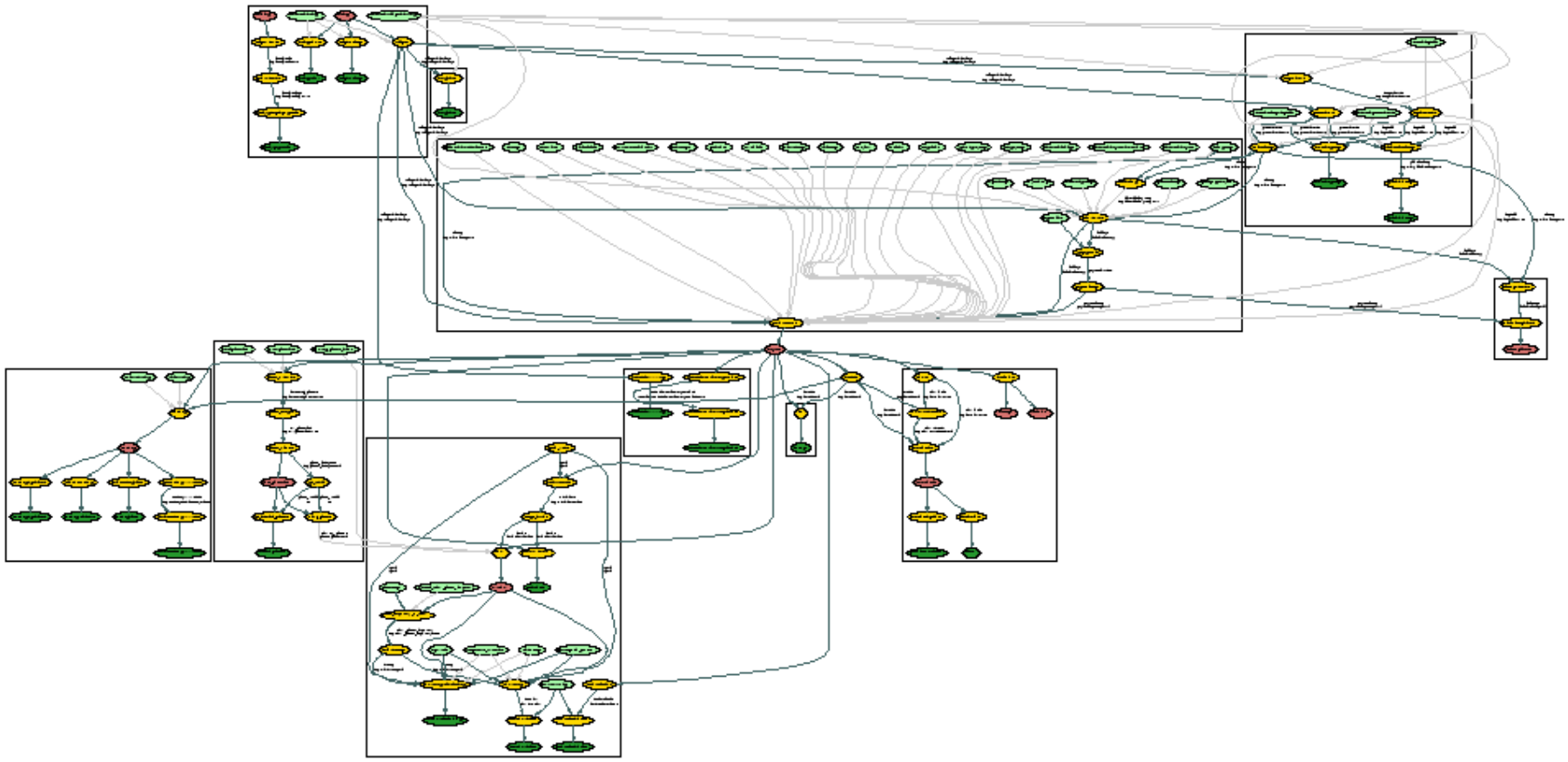


Final product: a map from each term to its compressed posting list

Each partition file contains docs with fixed (hash(url) % num files)

Google™

Real indexing pipelines are a bit messier…



Google™

PageRank

Indexing pipeline can answer the question

What are all the pages that match this query?

But most people don't want to look at all million hits. We need to answer

What are the best pages that match this query?

PageRank: use the link structure of the web to find “best” pages

PageRank

Suppose you browsed the web randomly, i.e. by clicking on random links.

What fraction of your time would you spend on each page?

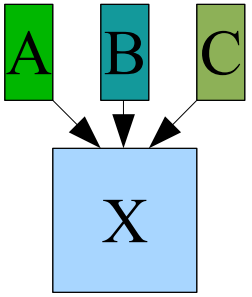
T(X) = time spent on page X

L(X) = # links from page X to other pages

Then the Ts and Ls would satisfy equations like

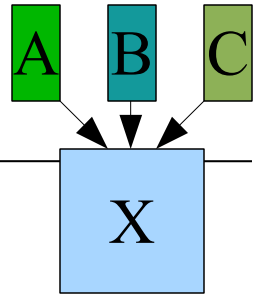
$$T(X) = T(A)/L(A) + T(B)/L(B) + T(C)/L(C)$$

if A, B, C are all the pages linking to X



Google™

PageRank



$$T(X) = T(A)/L(A) + T(B)/L(B) + T(C)/L(C)$$

Problems:

- Pages with no outbound links = black holes
- How to calculate? Is there even a unique solution?

Solution to both: damping factor (d = .85)

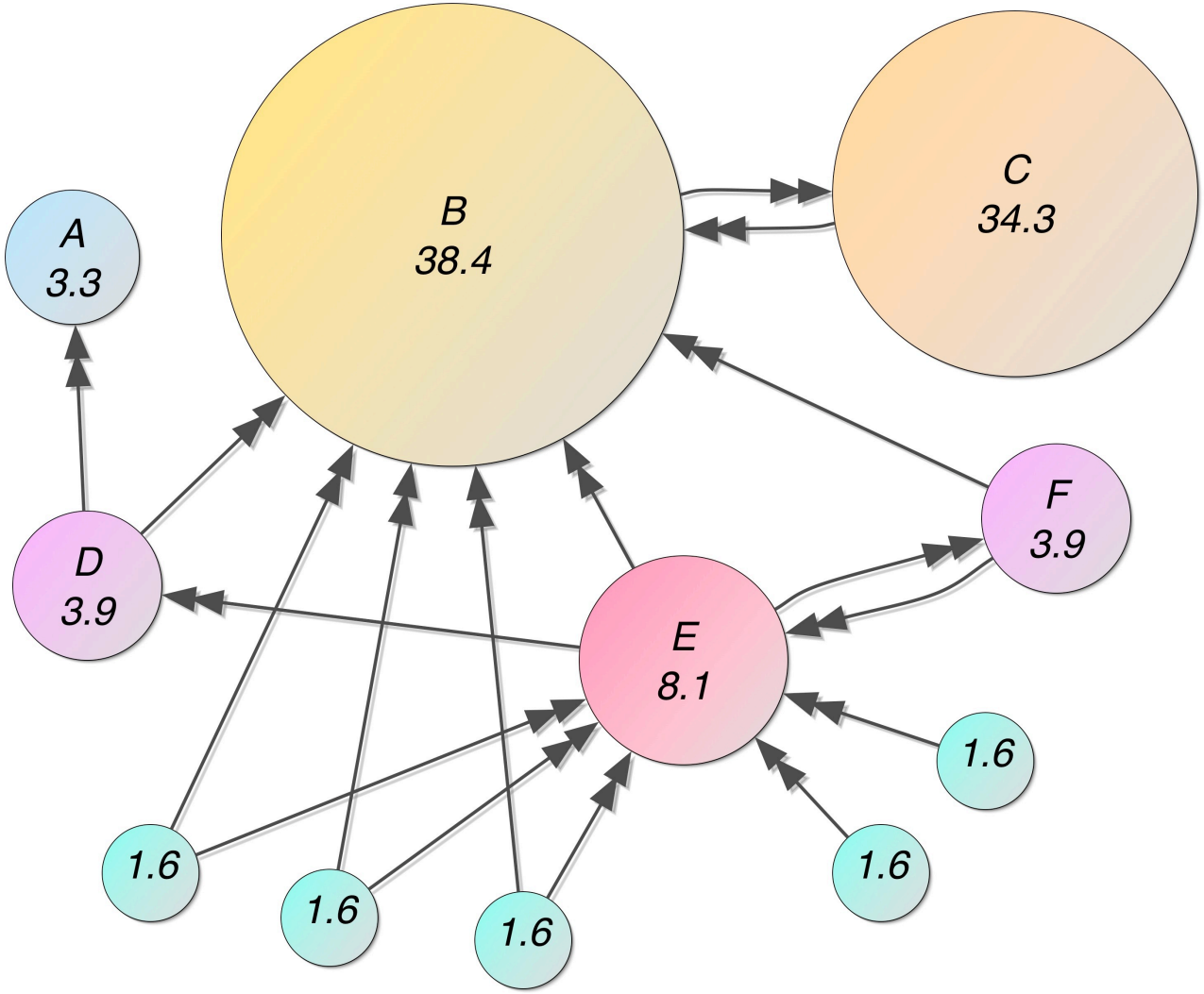
$$PR(X) = (1-d) + d*(PR(A)/L(A) + PR(B)/L(B) + PR(C)/L(C))$$

Random browsing, except with probability 1-d you jump to a random page.

Brin & Page, “The Anatomy of a Large-Scale Hypertextual Web Search Engine” (1998)

Google™

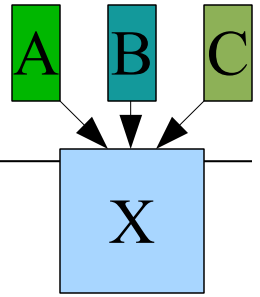
PageRank



Public domain image by Wikipedia user 345Kai, appears on http://en.wikipedia.org/wiki/PageRank

Google™

PageRank



$$PR(X) = (1-d) + d*(PR(A)/L(A) + PR(B)/L(B) + PR(C)/L(C))$$

We can calculate PageRank iteratively.
$PR_i(X)$ = probability of being on page X after *i* random steps

- Initial state:

$$PR_0(X) = 1 \text{ for all pages X.}$$

- Update:

$$PR_{i+1}(X) = (1-d) + d \sum_{Y \to X} PR_i(Y)/L(Y)$$

- $PR_i(X)$ converges to PR(X) as $i \to \infty$. Keep iterating.

Google

PageRank: MapReduce Implementation

$$PR(X) = (1-d) + d*(PR(A)/L(A) + PR(B)/L(B) + PR(C)/L(C))$$

- Perform one MR to produce the link graph and set up initial state

  (URL, contents) ↦ (URL, [1.0, list of linked-to URLs])

- Each MapReduce performs one step of the iteration
  The mapper and reducer keys are both the set of all URLs

  Map: $(Y, [PR(Y), Z_1...Z_n]) \mapsto (Z_i, PR(Y)/n)$ for i=1,...,n

  $(Y, [Z_1...Z_n])$ remember the link graph

  Reduce: $(Y, <S_0, S_1,...,S_m, [Z_1...Z_n]>) \mapsto (Y, [PR(Y), Z_1...Z_n])$

- An outside controller runs each MR iteration and decides when to stop.