DD2476: Search Engines and Information Retrieval Systems

Johan Boye*
KTH

Lecture 4

* Many slides inspired by Manning, Raghavan and Schütze

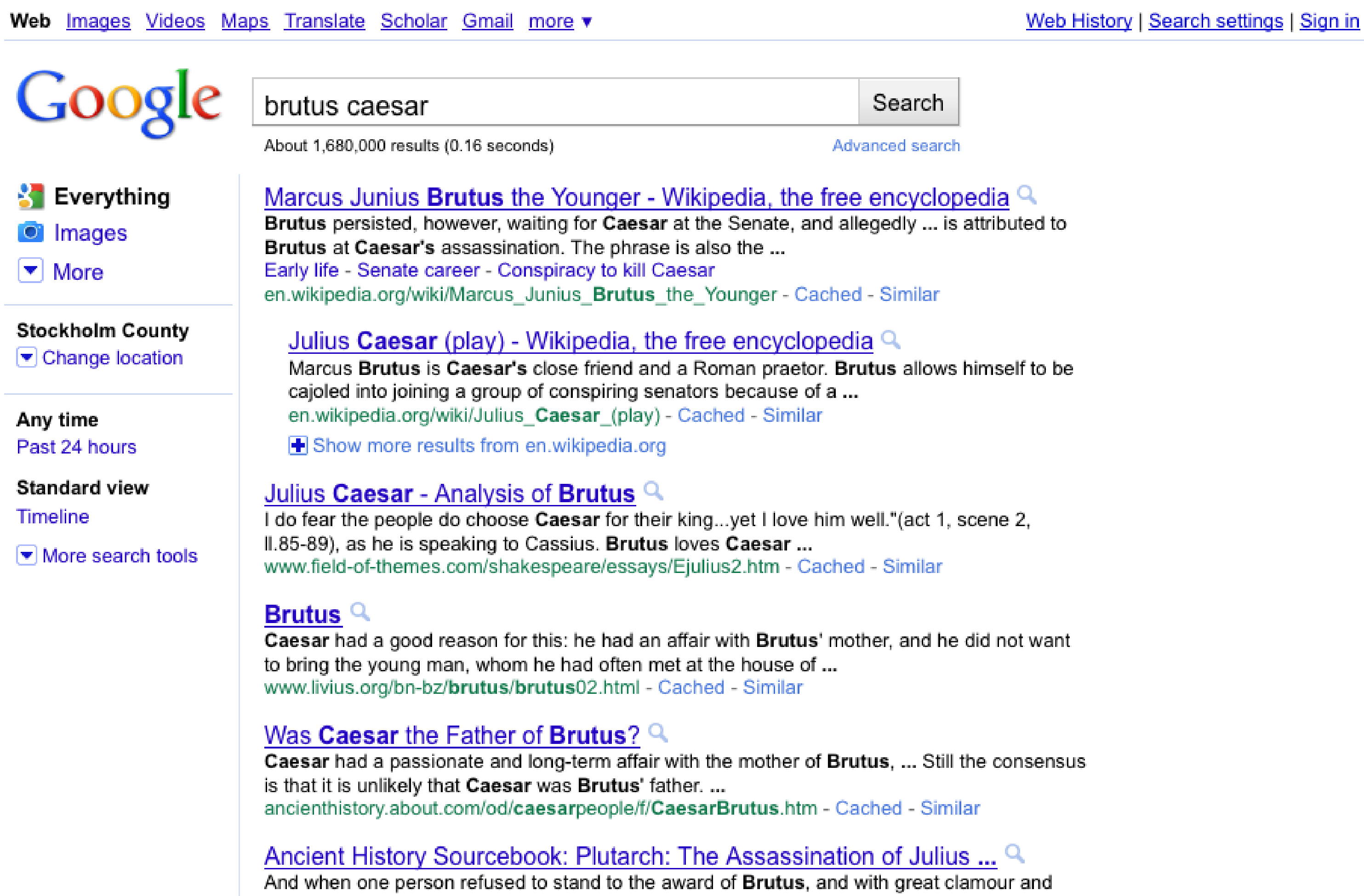
Remember: Boolean retrieval

- In computer assignment 1, you implemented a special case of Boolean retrieval (intersection).
- Boolean retrieval might be good for expert users
- But it is bad for most users, especially for web search.

Problems with Boolean search

- Boolean queries often return **too many** or **too few** results
  - ”zyxel P-660h” → 192 000 results
  - ”zyxel P-660h” ”no card found” → 0 results
- Takes skill to formulate a search query that gives a manageable number of hits.
  - ”AND” gives too few, ”OR” too many
- No ranking of search results

Ranked retrieval



Ranked retrieval

- Every matching document is given a score, say in [0..1]
- The **higher** the score, the **better** the match
- Large result sets do not pose problems
  - Show top *k* results ($k \approx 10$)
  - Option to see more.
  - **Premise**: The ranking algorithm works!

Today's topics

- The **vector space model**
  - documents and queries are represented as vectors in a high-dimensional space
- **tf_idf weighting**
  - take the frequency and informativeness of terms into account
- The PageRank algorithm
  - Static relevance scores through link analysis

Term-document incidence matrix

	Antony & Cleopatra	Julius Caesar	Tempest	Hamlet	Othello	Macbeth
Antony	1	1	0	0	0	1
Brutus	1	1	0	1	0	0
Caesar	1	1	0	1	1	1
Calpurnia	0	1	0	0	0	0
Cleopatra	1	0	0	0	0	0
mercy	1	0	0	1	1	1
citizen	1	1	0	0	1	0

1 if term is present in document, 0 otherwise

Word count matrix

	Antony & Cleopatra	Julius Caesar	Tempest	Hamlet	Othello	Macbeth
Antony	157	73	0	0	0	1
Brutus	4	157	0	1	0	0
Caesar	232	227	0	2	1	1
Calpurnia	0	10	0	0	0	0
Cleopatra	57	0	0	0	0	0
mercy	2	0	3	5	5	1
citizen	1	2	0	0	1	0

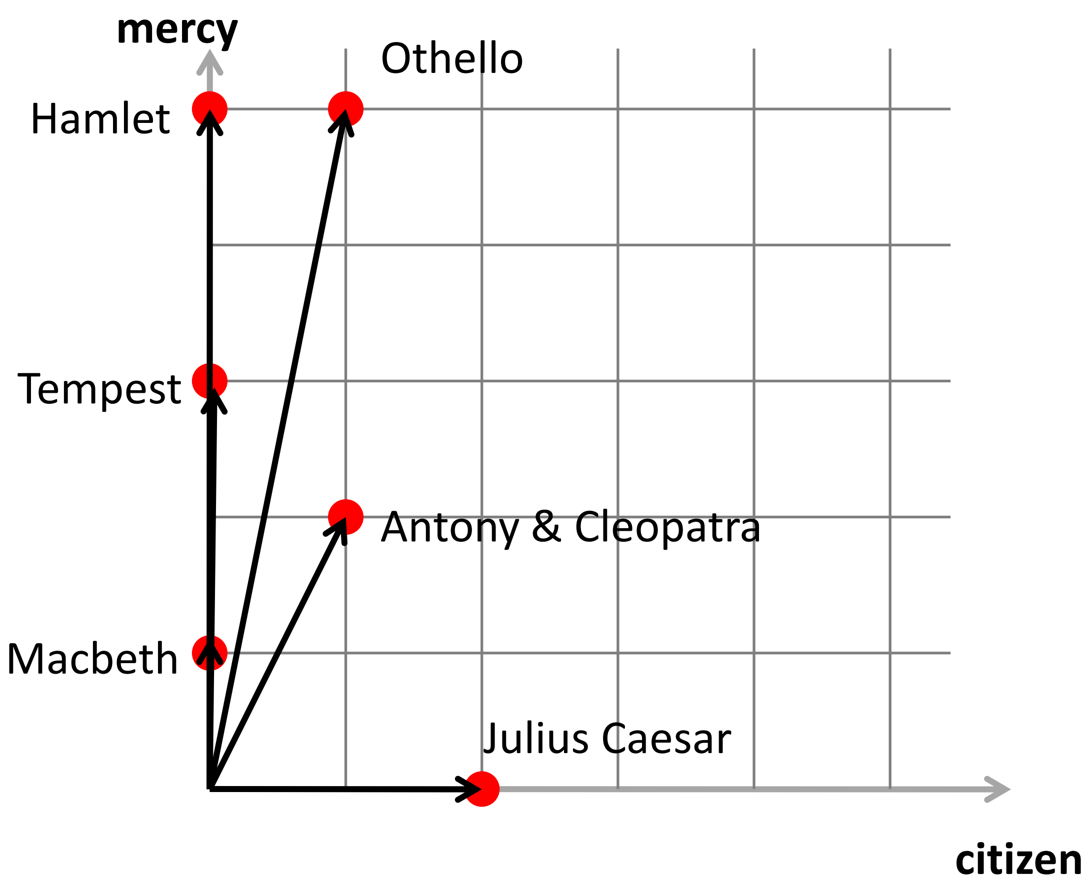
Every document is a vector in term space.

Word count matrix

	Antony & Cleopatra	Julius Caesar	Tempest	Hamlet	Othello	Macbeth
Antony	157	73	0	0	0	1
Brutus	4	157	0	1	0	0
Caesar	232	227	0	2	1	1
Calpurnia	0	10	0	0	0	0
Cleopatra	57	0	0	0	0	0
mercy	2	0	3	5	5	1
citizen	1	2	0	0	1	0

Let’s have a look at these dimensions only.

Documents as vectors



Bag-of-words

- Represent documents as vectors

$$d = (c_1, c_2, \ldots, c_n)$$

where c_i is the number of occurrences of word w_i

- Called a **bag-of-words** representation ('bag' = multiset)
- Ordering of words **not** considered
  - "Carl is wiser than Mary" and "Mary is wiser than Carl" have the **same** vector

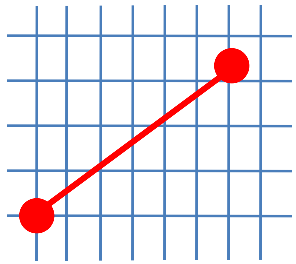
Documents as vectors

- So we have a |V|-dimensional space
  - Terms are axes/dimensions
  - Documents are points/vectors in this space
- Very high-dimensional
  - ~180,000 dimensions for our davisWiki corpus, much more for entire web
- Very sparse vectors - most entries zero

Comparing points (vectors)

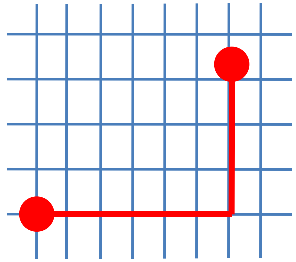
- **Euclidean** distance between $u = (u_1 \ldots u_n)$ and $v=(v_1 \ldots v_n)$

$$\sqrt{\sum_{i=1}^{n} (u_i - v_i)^2}$$

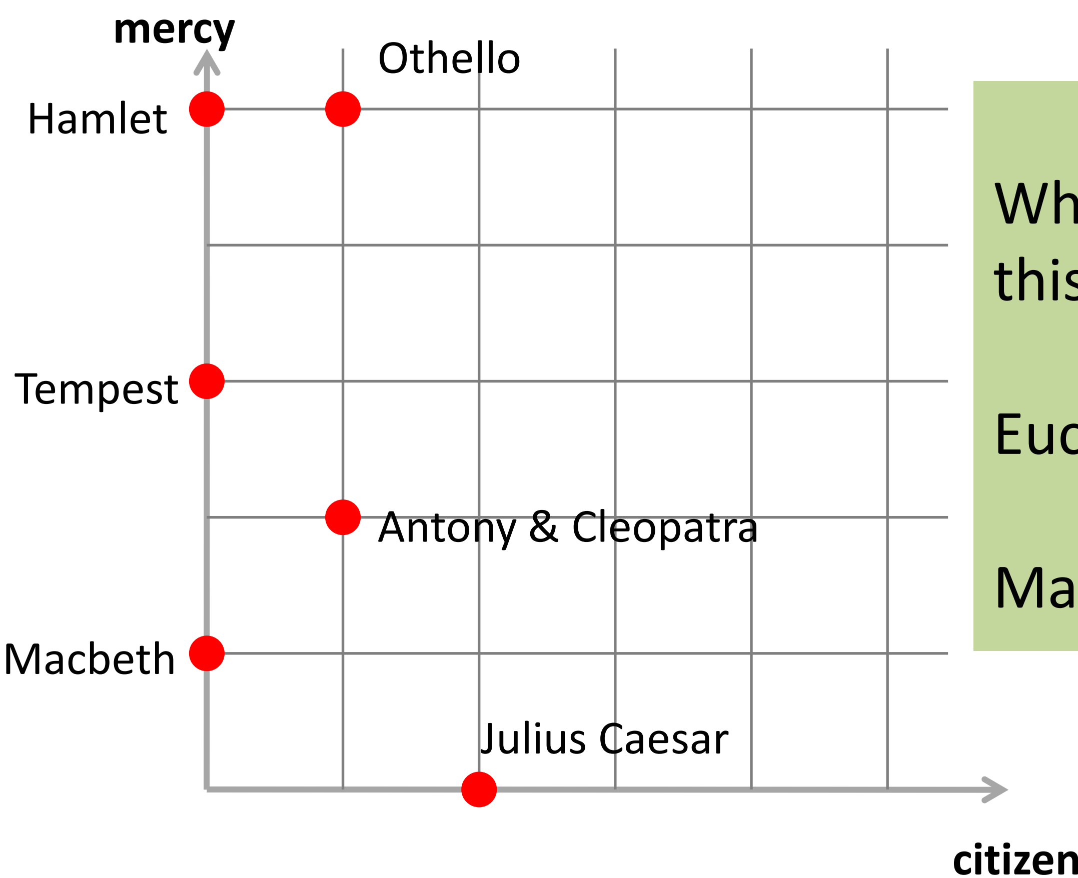


- **Manhattan** distance between u and v:

$$\sum_{i=1}^{n} \left| u_i - v_i \right|$$



Documents as vectors



What is d(Hamlet, A&C) in this space?

Euclidean: $\sqrt{(5-2)^2+(0-1)^2} = \sqrt{10}$

Manhattan: $|5-2|+|0-1| = 4$

What's the point?

- Suppose we have the query

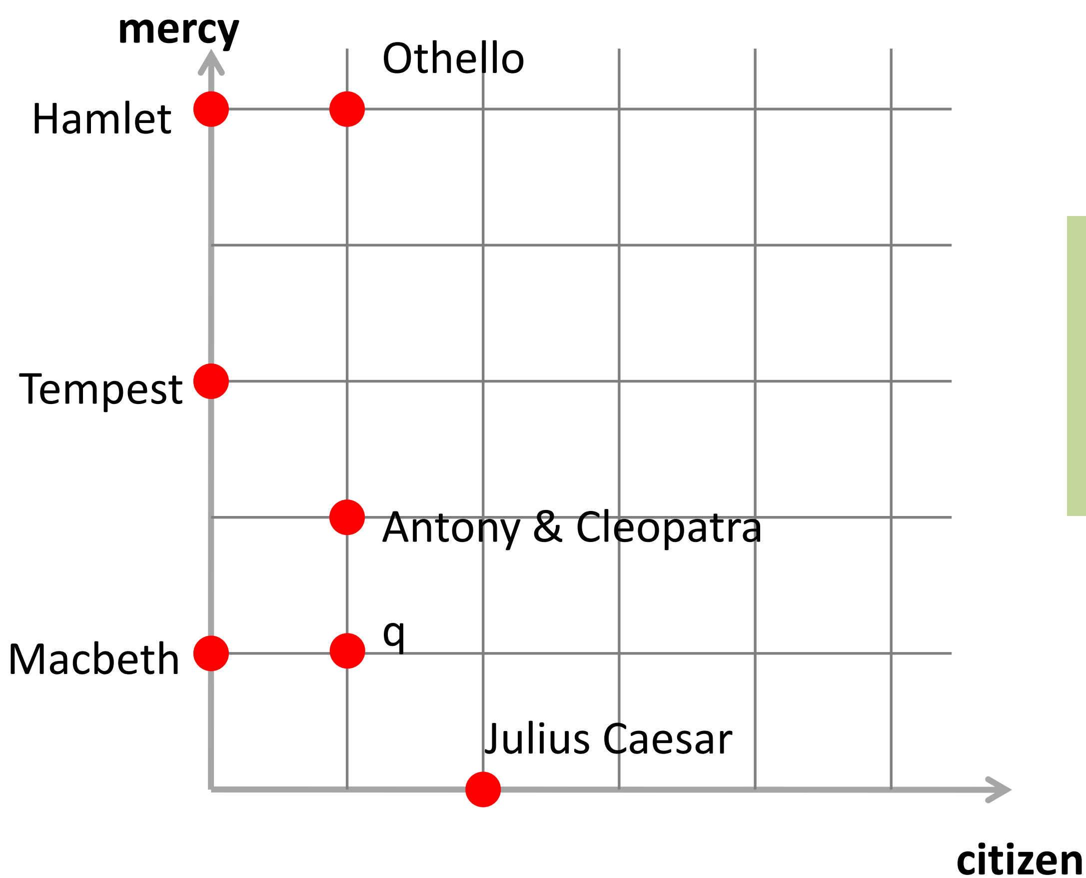
```
mercy citizen
```

- This query can be represented as the vector q=(1,1) (in the mercy-citizen space).
- Perhaps the most relevant documents are those closest to the query?

Queries as vectors

- **Key idea 1:** Represent queries as vectors in same space
- **Key idea 2:** Rank documents according to proximity to query in this space
- Recall:
  - Get away from Boolean model
  - Rank more relevant documents higher than less relevant documents

Documents as vectors



A&C and Macbeth are closest to q.

Short distance = high relevance?

- However consider the query

```
information retrieval
```

- and the 2 documents:
  - the Wikipedia article on Information Retrieval
  - “blue fish”
- Which is most relevant?
- Which is closest to the query?

Dot product

- Recall: the **dot product** of two vectors is

$$u \cdot v = \sum_{i=0}^{n} u_i v_i$$

- e.g. the dot product of “information retrieval” and “blue fish” will be 0.

”information retrieval”	(0,0,...,1,...0,...,0,...,1,...)
”blue fish”	(0,0,...,0,...1,...,1,...,0,...)

Dot product

$$u \cdot v = \sum_{i=0}^{n} u_i v_i$$

- The dot product of “information retrieval” and the Wikipedia article on IR will be large (=206)

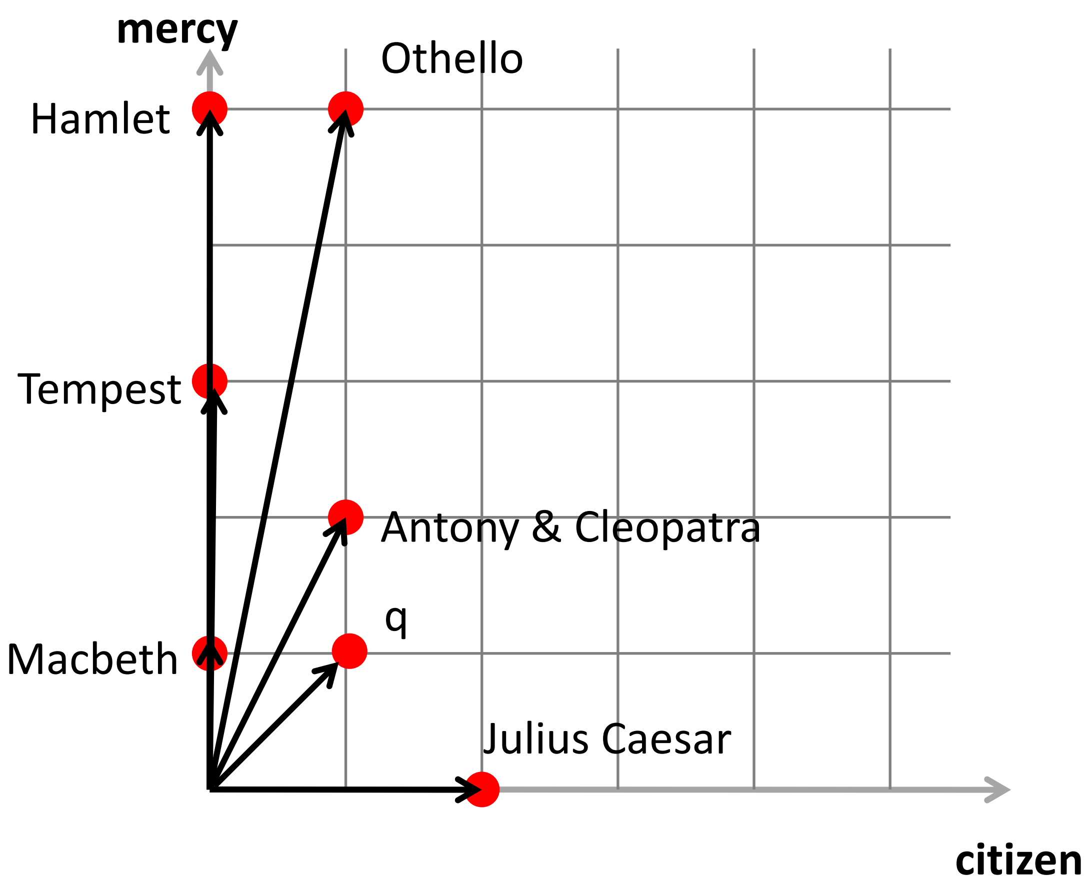
”information retrieval”	(0,0,...,1,............1,...)
the Wiki IR article	(0,0,..,103,.........93..)

Large dot product = high relevance?

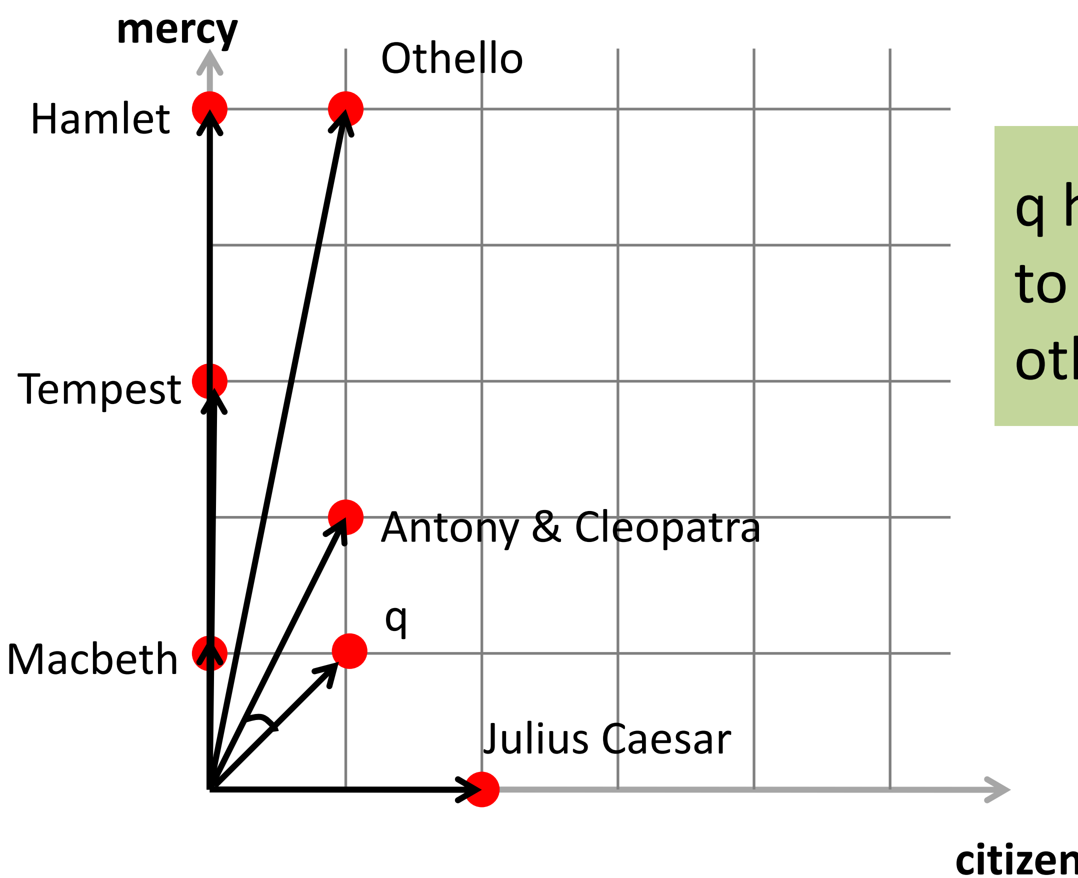
- So should we use the dot product as a rating mechanism?

- However, only using the dot product will favour long documents (why)?

Documents as vectors



Documents as vectors



q has a smaller angle to A&C is than to any other document.

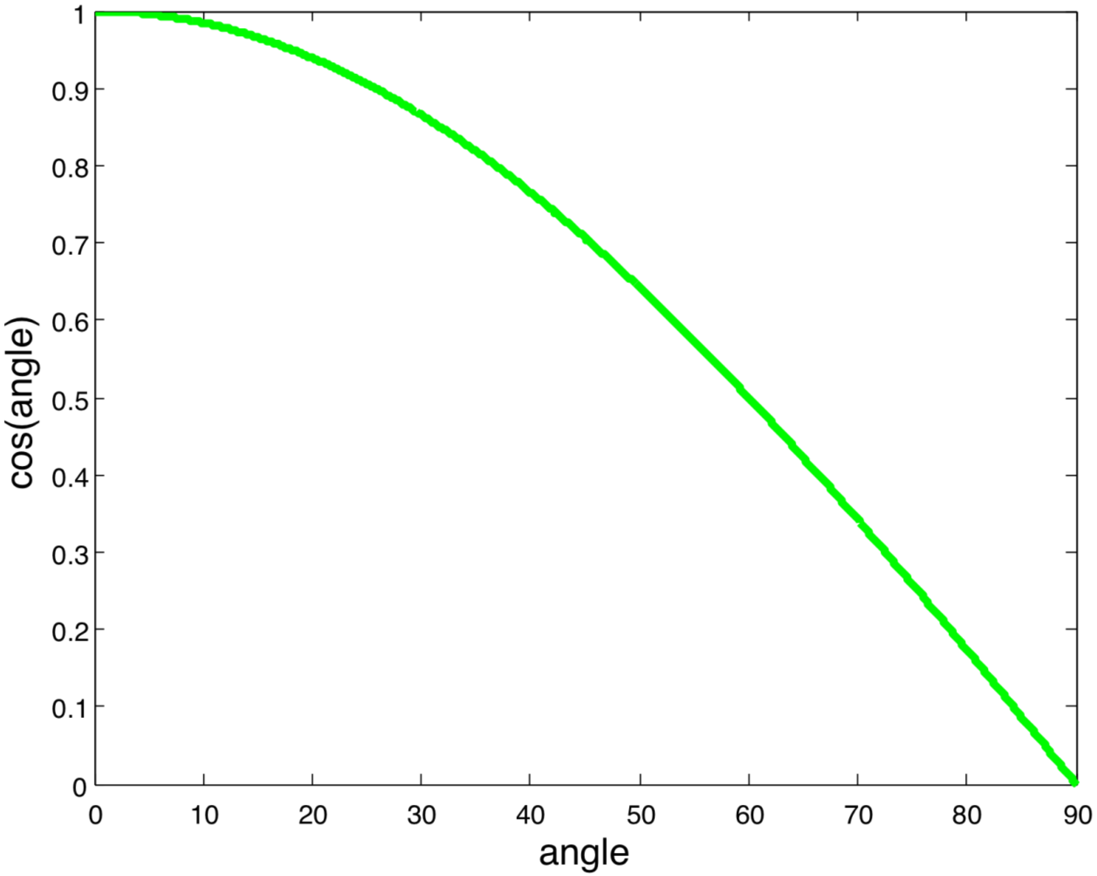
Small angle= high relevance?

- Small angle between two docs d1 and d2 = the **distribution of terms** is **similar** in d1 and d2

- So should we use the angle as a rating mechanism?
  - Small angle with q = higher rating?

- In fact, we will use the **cosine** of the angle (rather than the angle itself)

Graph of cos(angle)



Monotonically decreasing

Cosine similarity

Dot product of *u* and *v*:

$$u \cdot v = \sum_{i=0}^{n} u_i v_i$$

It holds that:

$$u \cdot v = \|u\| \|v\| \cos \theta$$

where |*u*| = the length of *u*, and θ = angle between *u* and *v*

Therefore:

$$\cos \theta = \frac{\sum_{i=0}^{n} u_i v_i}{\|u\| \|v\|}$$

Length of a vector

- There is more than one length norm:
  - Manhattan

$$\|x\|_1 = \sum_i |x_i|$$

  - Euclidean

$$\|x\|_2 = \sqrt{\sum_i x_i^2}$$

Cosine similarity

Dot product Unit vectors

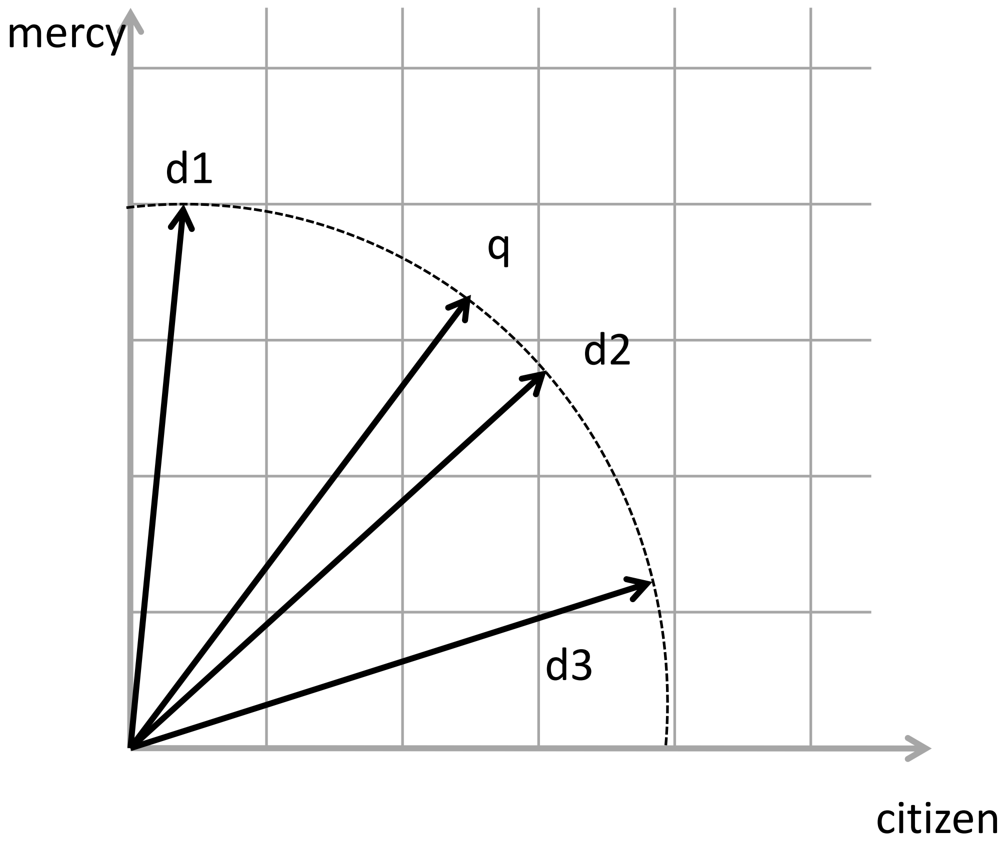
$$\cos(q,d) = \frac{q \cdot d}{|q||d|} = \frac{q}{|q|} \cdot \frac{d}{|d|} = \frac{\sum_{i=0}^{n} q_i d_i}{\sqrt{\sum_{i=0}^{n} q_i^2}\sqrt{\sum_{i=0}^{n} d_i^2}}$$

cos(q,d) = is the **cosine similarity** of *q* and *d*

= cosine of the angle between *q* and *d*

= dot product of the unit vectors *q*/|*q*| and *d*/|*d*|

Vectors after length normalisation



Word count matrix

	Antony & Cleopatra	Julius Caesar	Tempest	Hamlet	Othello	Macbeth
Antony	157	73	0	0	0	1
Brutus	4	157	0	1	0	0
Caesar	232	227	0	2	1	1
Calpurnia	0	10	0	0	0	0
Cleopatra	57	0	0	0	0	0
mercy	2	0	3	5	5	1
citizen	1	2	0	0	1	0

Term frequencies **tf**

log-frequency weighting

- Which numbers should fill our vectors?
- Raw term frequency might not be what we want:
  - A document with 10 occurrences of the term is more relevant than a document with 1 occurrence of the term
  - But perhaps not 10 times more relevant
- **Log-frequency weight** of term *t* in document *d*

$$w_{t,d} = \begin{cases} 1 + \log_{10} \mathrm{tf}_{t,d}, & \text{if } \mathrm{tf}_{t,d} > 0 \\ 0, & \text{otherwise} \end{cases}$$

Example

$$w_{t,d} = \begin{cases} 1 + \log_{10} \mathrm{tf}_{t,d}, & \text{if } \mathrm{tf}_{t,d} > 0 \\ 0, & \text{otherwise} \end{cases}$$

term	$tf_{t,d}$	$w_{t,d}$
airplane	0	
shakespeare	1	
calpurnia	10	
under	100	
the	1,000	

Document frequency *df*

- Rare terms are more informative than frequent terms
- Example: rare word CAPRICIOUS
  - Document containing this term is very likely to be relevant to query CAPRICIOUS
    → High weight for rare terms like CAPRICIOUS
- Example: common word THE
  - Document containing this term can be about anything
    → Very low weight for common terms like THE
- We will use **document frequency** (df) to capture this.

idf (inverse ***df***)

- Informativeness ***idf*** (inverse document frequency) of *t*:

$$\mathrm{idf}_t = \log_{10}(N/\mathrm{df}_t)$$

where N is the number of documents.

log (N/df_t) instead of N/df_t to "dampen" the effect of idf.

tf_idf weighting

- **tf_idf weight** of a term: product of tf weight and idf weight
- Best known weighting scheme in information retrieval
- Increases with the **number of occurrences** within a document
- Increases with the **rarity of the term** in the collection

Effect of idf on ranking

- Note that idf has no effect on ranking for one-term queries, like 'CAPRICIOUS'.

- Only effect for >1 term
  - Query THE CAPRICIOUS PERSON: idf puts more weight on CAPRICIOUS than PERSON…
  - … and much more than THE

Cosine similarity again

Dot product

Unit vectors

$$\cos(q,d)=\underbrace{\frac{q\cdot d}{|q||d|}}_{}=\frac{q}{|q|}\cdot\frac{d}{|d|}=\frac{\sum_{i=0}^{n} q_i d_i}{\sqrt{\sum_{i=0}^{n} q_i^2}\sqrt{\sum_{i=0}^{n} d_i^2}}$$

q_i is the tf-idf weight of term *i* in the query

d_i is the tf-idf weight of term *i* in the document

tf_idf-weighting - Example

d1	to be or not to be
d2	to be is to do
d3	i do i do i do i do i do
d4	do be do be do

- What is idf(to)?
- What is tf(d1,to)?
- What do the d1-d4 vectors look like?
- What is cos(d1, d2)?

Computing cosine scores

COSINESCORE(q)

float $Scores[N] = 0$
float $Length[N]$
for each query term t
do calculate $w_{t,q}$ and fetch postings list for t
  for each pair$(d, \mathrm{tf}_{t,d})$ in postings list
  do $Scores[d] += w_{t,d} \times w_{t,q}$
Read the array *Length*
for each d
do $Scores[d] = Scores[d]/Length[d]$
return Top K components of $Scores[]$

Computing cosine scores

- In the code skeleton for the assignments...
- ... in the **`Index.java`** interface...
- ... there is a HashMap **`docLengths`** that stores the number of tokens (=Manhattan length) for all documents.
- This is computed for you at indexing time.

Weighting schemes

- Different weighting schemes:

Term frequency		Document frequency		Normalization	
n (natural)	$\mathrm{tf}_{t,d}$	n (no)	1	n (none)	1
l (logarithm)	$1 + \log(\mathrm{tf}_{t,d})$	t (idf)	$\log \frac{N}{\mathrm{df}_t}$	c (cosine)	$\frac{1}{\sqrt{w_1^2 + w_2^2 + \ldots + w_M^2}}$
a (augmented)	$0.5 + \frac{0.5 \times \mathrm{tf}_{t,d}}{\max_t(\mathrm{tf}_{t,d})}$	p (prob idf)	$\max\{0, \log \frac{N - \mathrm{df}_t}{\mathrm{df}_t}\}$	u (pivoted unique)	$1/u$ (Section 6.4.4)
b (boolean)	$\begin{cases} 1 & \text{if } \mathrm{tf}_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases}$			b (byte size)	$1/\mathit{CharLength}^{\alpha}$, $\alpha < 1$
L (log ave)	$\frac{1 + \log(\mathrm{tf}_{t,d})}{1 + \log(\mathrm{ave}_{t \in d}(\mathrm{tf}_{t,d}))}$				

- In assignment 2.3 you will explore some of these variants

Summary – Vector space model

- Vector space model:
  - Represent the query as a tf-idf vector
  - Represent each document as a tf-idf vector
  - Compute the cosine similarity score for the query vector and each document vector
  - Rank documents with respect to the query by score
  - Return the top K (e.g., K = 10) to the user
- In assignment 2.1-2.2 you will extend your search engine to do the above

Computing cosine scores efficiently

- Approximation:
  - Assume that terms only occur once in query document

$$w_{t,q} \leftarrow \begin{cases} 1, & \text{if } \mathrm{w}_{t,q} > 0 \\ 0, & \text{otherwise} \end{cases}$$

- Works for short documents ($|d| << N$)
- Works since ranking only relative

Computing cosine scores efficiently

FastCosineScore(q)

float $Scores[N] = 0$
for each d
do Initialize $Length[d]$ to the length of doc d
for each query term t
do calculate $w_{t,q}$ and fetch postings list for t
  for each pair(d, $tf_{t,d}$) in postings list
  do add $wf_{t,d}$ to $Scores[d]$
Read the array $Length[d]$
for each d
do Divide $Scores[d]$ by $Length[d]$
return Top K components of $Scores[]$

Figure 7.1 A faster algorithm for vector space scores.

Computing cosine scores efficiently

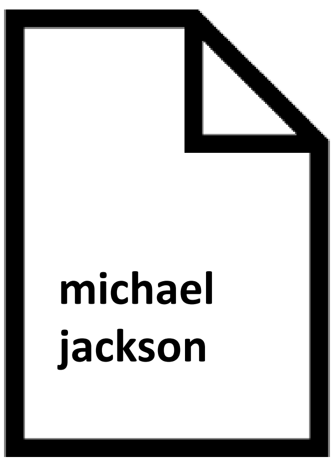
- Downside of approximation: **sometimes get it wrong**
  - A document not in the top *K* may creep into the list of *K* output documents

- Is this such a bad thing?

- Cosine similarity is only a proxy
  - User has a task and a query formulation
  - Cosine matches documents to query
  - Thus cosine is anyway a proxy for user happiness
  - If we get a list of *K* documents “close” to the top *K* by cosine measure, should be ok

A problem with tf_idf ranking

michael jackson



Better cosine similarity with the query!

Solution: Static quality scores

- We want top-ranking documents to be both **relevant** and **authoritative**
  - Relevance – cosine scores
  - Authority – query-independent property

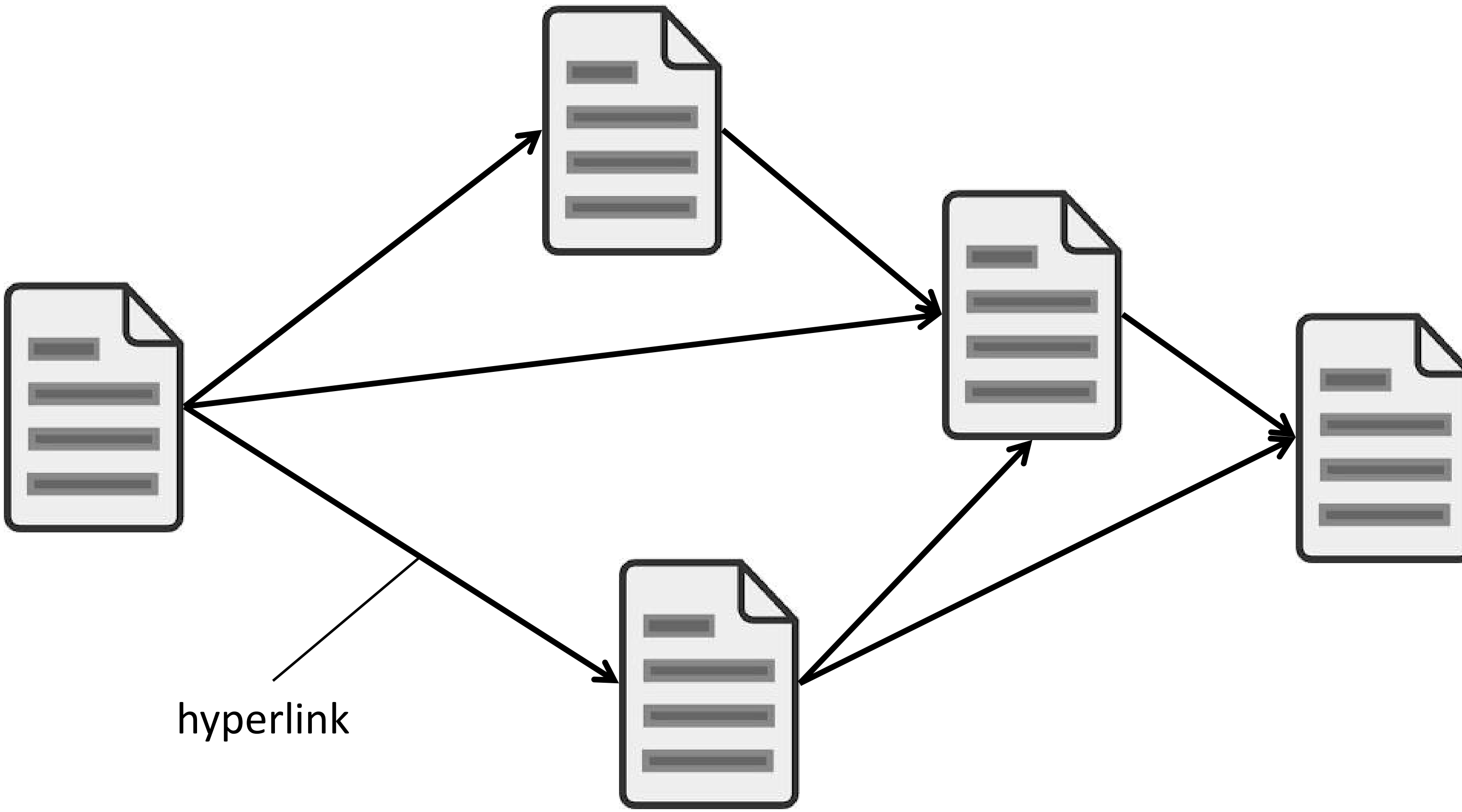
- Assign **query-independent quality score** g(*d*) in [0,1] to each document *d*

- $\text{net-score}(q,d) = w_1 * g(d) + w_2 * \cos(q,d)$
  - Two “signals” of user happiness

Static quality scores

- Which pages should be highly ranked? Alternatives:
  - Personalised search: **Pages I've visited before**
  - Pages **visited by lots of users**
  - **Well-known quality** pages (Wikipedia, NY Times, …)
  - Pages with **many important in-links** (PageRank)
  - Money talks: **Sponsored links**
  - (and recently): Veracity: **Pages with true information** are ranked highly, alternative facts less so.

The Web as a directed graph



Using link structure for ranking

- **Assumption:** A link from X to Y signals that X's author perceives Y to be an authoritative page.
  - X "casts a vote" on Y.
- **Simple suggestion:** Rank = number of in-links
- However, there are problems with this naive approach.

PageRank: basic ideas

- WWW’s particular structure can be exploited
  - pages have links to one another
  - the more in-links, the higher rank
  - in-links from pages having **high rank** are **worth more** than links from pages having low rank
  - this idea is the cornerstone of **PageRank** (Brin & Page 1998)

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

PageRank – first attempt

$$PR(p) = \sum_{q \in in(p)} \frac{PR(q)}{L_q}$$

- *p* and *q* are pages
- *in(p)* is the set of pages linking to *p*
- L_q is the number of out-links from *q*

The random surfer model

- Imagine a **random surfer** that follow links
- The link to follow is selected with uniform probability
- If the surfer reaches a **sink** (a page without links), he **randomly restarts** on a new page
- Every once in a while, the surfer **jumps to a random page** (even if there are links to follow)

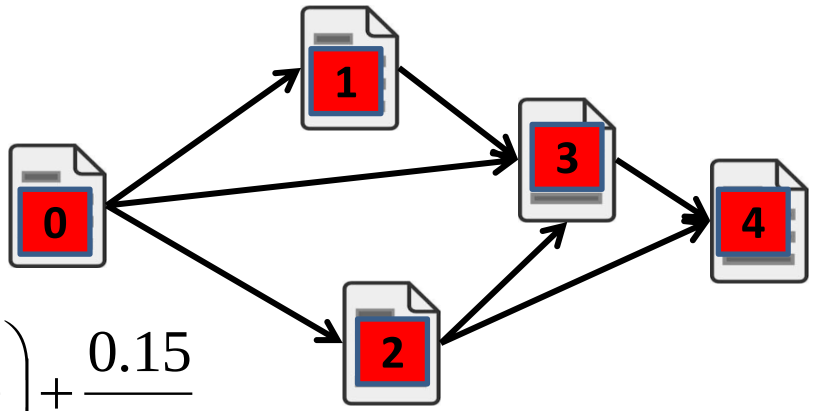
PageRank – second attempt

- With **probability *1-c*** the surfer is bored, **stops following links**, and **restarts** on a random page
- Guess: Google uses *c*=0.85

$$PR(p) = c\left(\sum_{q \in in(p)} \frac{PR(q)}{L_q} \right) + \frac{(1-c)}{N}$$

- Without this assumption, the surfer will get stuck in a subset of the web.

PageRank example



$$PR_4 = 0.85 \cdot \left(\frac{PR_2}{2} + PR_3 \right) + \frac{0.15}{5}$$

$$PR_3 = 0.85 \cdot \left(\frac{PR_0}{3} + PR_1 + \frac{PR_2}{2} + \frac{PR_4}{5} \right) + \frac{0.15}{5}$$

$$PR_2 = PR_1 = 0.85 \cdot \left(\frac{PR_0}{3} + \frac{PR_4}{5} \right) + \frac{0.15}{5}$$

$$PR_0 = 0.85 \cdot \left(\frac{PR_4}{5} \right) + \frac{0.15}{5}$$