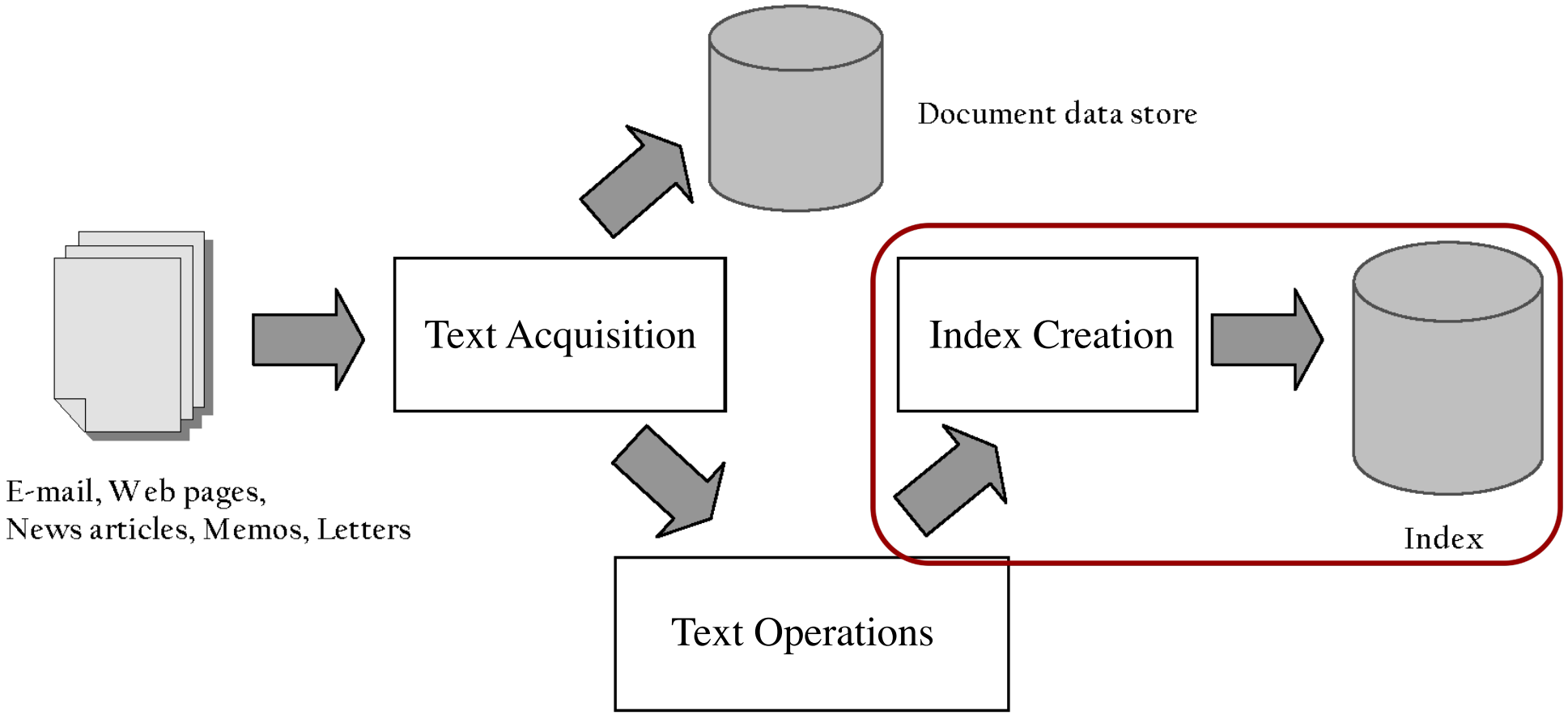
Indexing Construction and Index Compression

Uke 41 - Lecture 8

Aims

- To learn how index construction and compression techniques work

Indexing


Document data store
Text Acquisition
Index Creation
Index
E-mail, Web pages,
News articles, Memos, Letters
Text Operations

Inverted files

Inverted files

Structure with two elements:

1. **Vocabulary**: the set of all different words in the text
2. **Occurrence**: the positions of words in the text

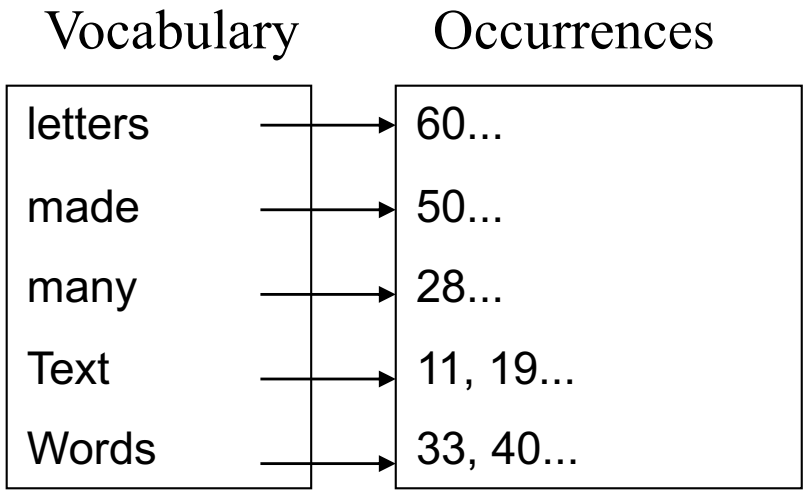
Example

- Text:

1	6	9	11	17	19	24	28	33	40	46	50	55	60

This is a text. A text has many words. Words are made from letters.

- Inverted file



Inverted files (3)

- Positions can be referred to words or characters
  - With **word positions**: simplification of proximity queries
  - With **character positions**: easy direct access to matching text positions

Inverted files (3)

- Low space requirement for the vocabulary
- More space requirement for the occurrence:
  - Each word appearing in the text is referenced once
  - Space overhead: 30% ~ 40% of the text size

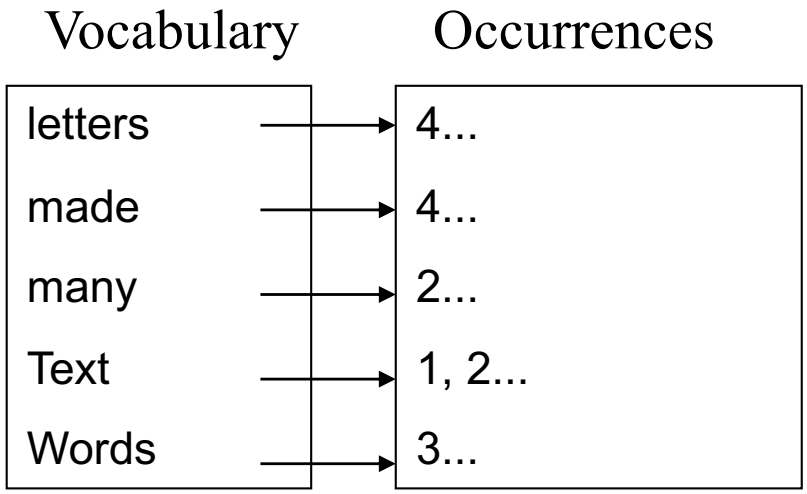
Inverted files - Block addressing

- Using block addressing to reduce the space requirement
  - Divide text into blocks
  - Occurrence points to the blocks
  - Cannot be used for proximity queries
    - Exact position required!

Inverted files - Block addressing (2)

Block 1	Block 2	Block 3	Block 4
This is a text.	A text has many	words. Words are	made from letters.

Text



Inverted index

Inverted files: Searching

Three general steps:

1. **Vocabulary search**: words in queries are isolated and searched separately
2. **Retrieval of occurrences**: retrieving occurrence of all the words
3. **Manipulation of occurrences**: occurrences processed to solve phrases, proximity or Boolean operations

Inverted files: Searching (2)

- Single word queries
  - Searched by Hashing, tries, or B-trees
- Prefix and range queries
  - Searched by binary search, tries or B-trees

Inverted files - Construction

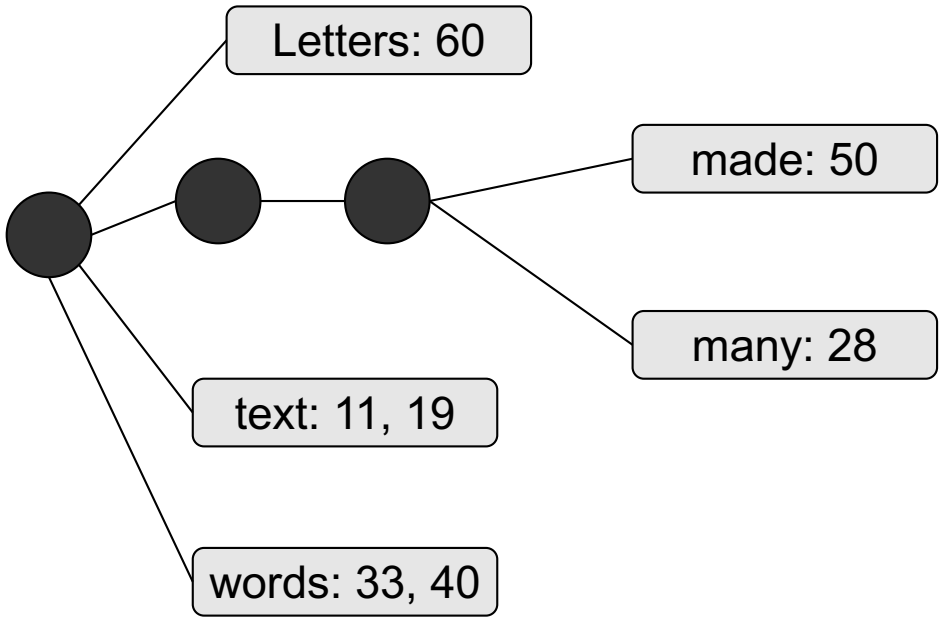
- **Trie** written to the disk together with the list of occurrences
- Index - two files:
  - First file: containing the **list of occurrences**
  - Second file: containing **vocabulary**, stored in lexicographical order
- Each word has a pointer to its list in the first file

Inverted files - Construction

1	6	9	11	17	19	24	28	33	40	46	50	55	60

This is a text. A text has many words. Words are made from letters.

Text



Vocabulary trie

Inverted files - Costs

- The total time to generate partial indices is ***O(n)***
- The number of partial indices is ***O(n/M)***
- To merge the ***O(n/M)*** partial indices are necessary ***$\log_2(n/M)$*** merging levels
- The total cost of this algorithm is ***O(n log(n/M))***

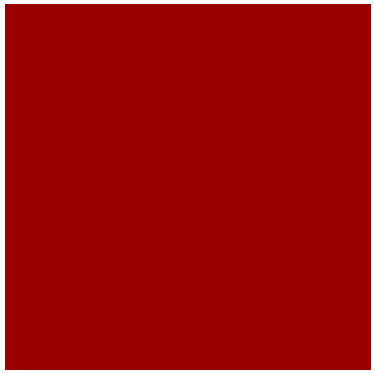
Inverted files - weaknesses

- Inverted files indices of only head of words
- Difficult to find suffixes and prefixes
- Expensive to handle

Inverted Index with posting

- Each **index term** associated with an **inverted list**
  - Contains lists of documents, or lists of **word occurrences** in documents, and other information
  - Each entry called a **posting**
  - The part of the posting referring to a specific document or location called a **pointer**
  - Each document in the collection given a unique number
  - Lists usually **document-ordered** (sorted by document number)

Example Collection



S_1 Tropical fish include fish found in tropical environments around the world, including both freshwater and salt water species.

S_2 Fishkeepers often use the term tropical fish to refer only those requiring fresh water, with saltwater tropical fish referred to as marine fish.

S_3 Tropical fish are popular aquarium fish, due to their often bright coloration.

S_4 In freshwater fish, this coloration typically derives from iridescence, while salt water fish are generally pigmented.

Four sentences from the Wikipedia entry for *tropical fish*

Example Collection (2)

and	1
aquarium	3
are	3 4
around	1
as	2
both	1
bright	3
coloration	3 4
derives	4
due	3
environments	1
fish	1 2 3 4
fishkeepers	2
found	1
fresh	2
freshwater	1 4
from	4
generally	4
in	1 4
include	1
including	1
iridescence	4
marine	2
often	2 3
only	2
pigmented	4
popular	3
refer	2
referred	2
requiring	2
salt	1 4
saltwater	2
species	1
term	2
the	1 2
their	3
this	4
those	2
to	2 3
tropical	1 2 3
typically	4
use	2
water	1 2 4
while	4
with	2
world	1

and	1:1
aquarium	3:1
are	3:1 4:1
around	1:1
as	2:1
both	1:1
bright	3:1
coloration	3:1 4:1
derives	4:1
due	3:1
environments	1:1
fish	1:2 2:3 3:2 4:2
fishkeepers	2:1
found	1:1
fresh	2:1
freshwater	1:1 4:1
from	4:1
generally	4:1
in	1:1 4:1
include	1:1
including	1:1
iridescence	4:1
marine	2:1
often	2:1 3:1
only	2:1
pigmented	4:1
popular	3:1
refer	2:1
referred	2:1
requiring	2:1
salt	1:1 4:1
saltwater	2:1
species	1:1
term	2:1
the	1:1 2:1
their	3:1
this	4:1
those	2:1
to	2:2 3:1
tropical	1:2 2:2 3:1
typically	4:1
use	2:1
water	1:1 2:1 4:1
while	4:1
with	2:1
world	1:1

Suffix trees and Suffix Arrays

Suffix trees and Suffix Arrays

- Text indexing data structures
- Suffix tree:
  - A **trie data structure** built over all the suffixes of he text
- Suffix array:
  - Indices of all the suffixes of a string

Suffix trees and Suffix Arrays (2)

- Viewing **texts as one (single) long string**
- All text positions uniquely identified by its position known as **index point**

Suffix trees and Suffix Arrays (3)

1	6	9	11	17	19	24	28	33	40	46	50	55	60

This is a text. A text has many words. Words are made from letters.

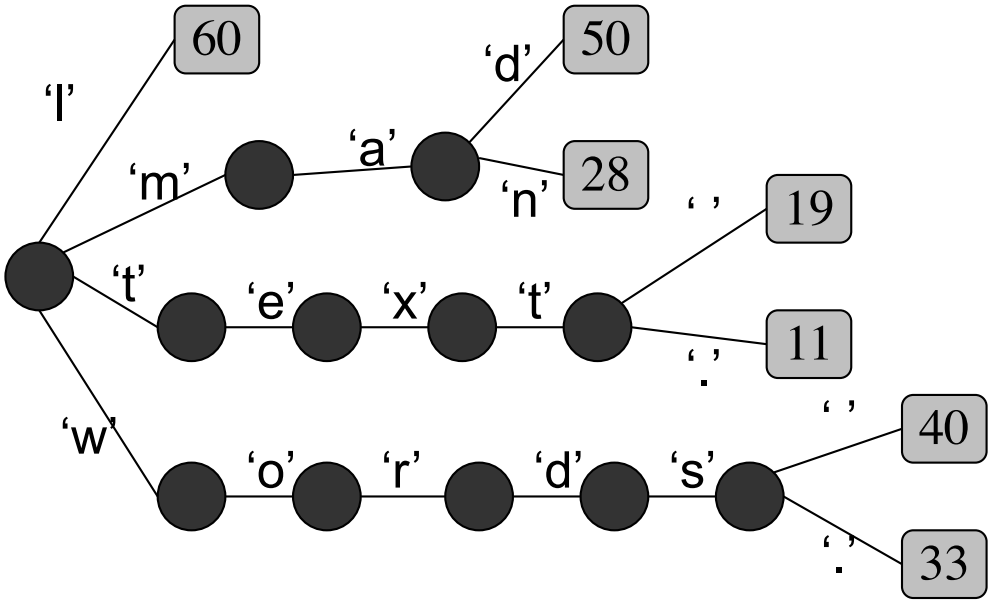
Text

String **11**: text. A text has many words. Words are made from letters.
String **19**: text has many words. Words are made from letters.
String **28**: many words. Words are made from letters.
String **33**: words. Words are made from letters.
String **40**: Words are made from letters.
String **50**: made from letters.
String **60**: letters.

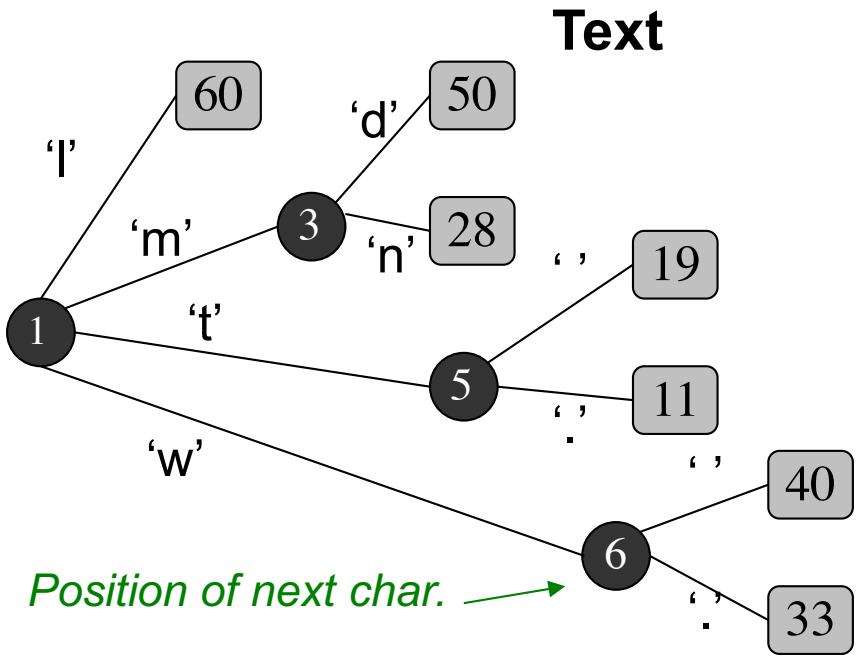
Suffixes

From Suffix tries to Suffix trees

1	6	9	11	17	19	24	28	33	40	46	50	55	60
This	is	a	text.	A	text	has	many	words.	Words	are	made	from	letters.



Suffix trie



Suffix tree

Suffix trees

Search in tries

- Traverse from root to leaf
- **A leaf reached**: successful search
- **No corresponding link found** an internal node: failed search

Suffix trees (2)

- Suffix trees have space problem
  - Each node takes 12 to 24 bytes
  - Space overhead: 120% ~ 240% over the text size

- Suffix arrays are better than suffix trees

Suffix arrays

- Same functionality as suffix trees but less space requirement
- Suffix array:
  - Array having all the pointers to the text suffixes listed in lexicographical order.
  - Concatenating all the pointers

Suffix arrays (2)

1	6	9	11	17	19	24	28	33	40	46	50	55	60

This is a text. A text has many words. Words are made from letters.

Text

Suffix array

60	50	28	19	11	40	33

Suffix arrays (3)

- Store one pointer per index suffix
- Space overhead: 40% over the text size
- Allowing binary searches done by comparing the contents of each pointer
  - Large array: **high number random disk** access ⇒ poor performance

Construction

- Suffix tree
  - Build in ***O(n)*** time (i.e. text of *n* characters)
  - Poor performance if suffix tree doesn't fit in main memory
- Suffix array
  - Build in ***O(nlog(n))*** time
  - A set of pointers, lexicographical sorted

Suffix trees and suffix arrays

Pros

- Efficient in searching more complex queries
  - The query can be any substring of the text

Cons

- Costly construction process
- Not suitable for approximate text search
- Results not delivered in text position order, but in a lexicographical order

Signature Files

Signature Files

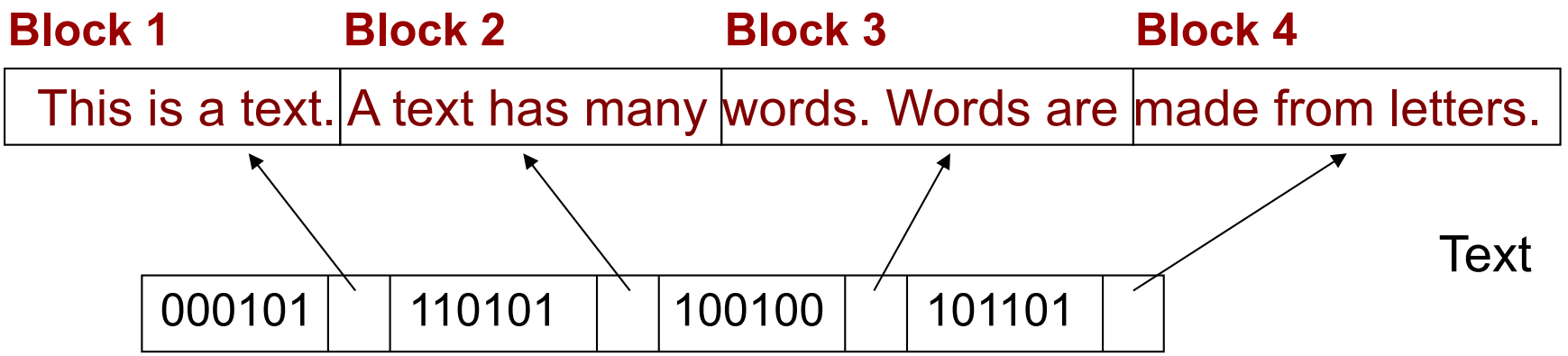
- Word-oriented
- Use hashing
- Low overhead: 10% to 20% of the text size
- Have to do sequential search
- Suitable for small texts

Signature files

- Word-oriented index structures based on hashing
- Signature (hash function)
  - Divide the text in blocks of b words each
  - Maps words to bit masks of B bits (signature)
  - Assign a bit mask of size B to each text block of size b
  - The mask is obtained by bitwise ORing the signatures of all the words in the text block

Signature files (2)

- Signature file
  - Collection of the bit masks of all blocks
  - Plus the pointers from the masks to blocks



h(text) = 000101
h(many) = 110000
h(words) = 100100
h(letters) = 100001

Signature function

Signature files (3)

- Idea
  - If a word is present in a text block, then all the bits set in the signature are also set in the bit masks of the text block
    - e.g., h('text')=000**1**0**1**, signature(block2)=110**1**0**1**
  - Whenever a bit is set in the mask of the query word and not in the mask of the text block, the the word is not present in the text block
    - e.g., h('text')=00010**1**, signature(block3)=10010**0**

Signature Files – False Drop

- Definition
  - All the corresponding bits in the mask are set even though the word is not there
- Design goal of signature file:
  - Low probability of a false drop
  - Short signature file (size)

Signature Files – Searching

- Procedure
  1. Hash a word to get a bit mask *W*
  2. Compare *W* with each bit masks B_i of all the text blocks
  3. If $W \,\&\, B_i = W$, the block of B_i may contain the word
  4. Collect all the candidate blocks
  5. Perform the sequential search for each of the candidate blocks

Signature Files – Searching (2)

- How about phrase and proximity queries
  - Very efficient
  - First, get the bit-wise OR of all the words in the query
    - All their bits must be present
  - Then perform the comparison with B_i's
- Performance of search
  - Quite slow
    - Experiment: 250Mb of text took 12 seconds

Signature Files - Construction

- Procedure
  - The text is cut in blocks
  - For each block, build the **bit-wise OR** of the signatures of all the words
  - Adding text
    - Add records into the signature file
  - Deleting text
    - Delete records from the signature file

Information Theory

Information Theory

- Entropy:
  - Measure for the amount of information in a given text
  - Concept to capturing information content(information uncertainty)

$$E = -\sum_{i=1}^{\sigma} p_i \log_2 p_i$$

- Text models
  - Affecting the amount of information measured in a text
  -

Modeling Natural Language

- Symbols
  - symbols separating word
  - symbols belonging to words

- Characteristic of symbol
  - not uniformly distributed
  - NL: having a dependency on previous symbols

Modeling Natural Language

- Issues for modeling Natural Language
  - Model to generate text
  - **Distribution of word frequency** in a doc (**Zipf's Law**)
  - Word distribution in docs
  - The **number of distinct words** in a doc (**Heap's Law**)
  - Average length of words

Modeling Natural Language (3)

- Models to generate a text
  - **binomial model**: simple model with each symbol generated with certain probability
  - **finite-context(Markovian) model**: model reflecting dependency of natural language
  - **finite-state model**: model defining regular languages
  - **grammar model**: model defining context free and other languages

Zipf's Law

- Approximate model for distribution of word frequencies
  - Frequency of i-th most frequent word: $1/i^{\theta}$ times the most frequent word
  - Frequency of j-th most frequent word in a text of *n* words with a vocabulary of *V* words:

$$\frac{n}{i^{\theta} Hv(\theta)} \qquad Hv(\theta) = \sum_{j=1}^{V} \frac{1}{j^{\theta}} = n$$

Harmonic number of order θ of V

Zipf's Law (2)

- The variation of θ depending in the text
  - if $\theta = 1 \Rightarrow H_v(\theta) = O(\log n)$
  - if $1.5 < \theta < 2.0 \Rightarrow H_v(\theta) = O(1)$: fitting better the real data

- Mandelbrot distribution:

$$Hv(\theta) = \frac{k}{(c+i)^{\theta}}$$

the value such that all frequency add to n

where, c > 0, 0 ≤ i < 1

- Reducing the space overhead of indices for text by disregarding stopword

Alternative Computation

Zipf's Law:

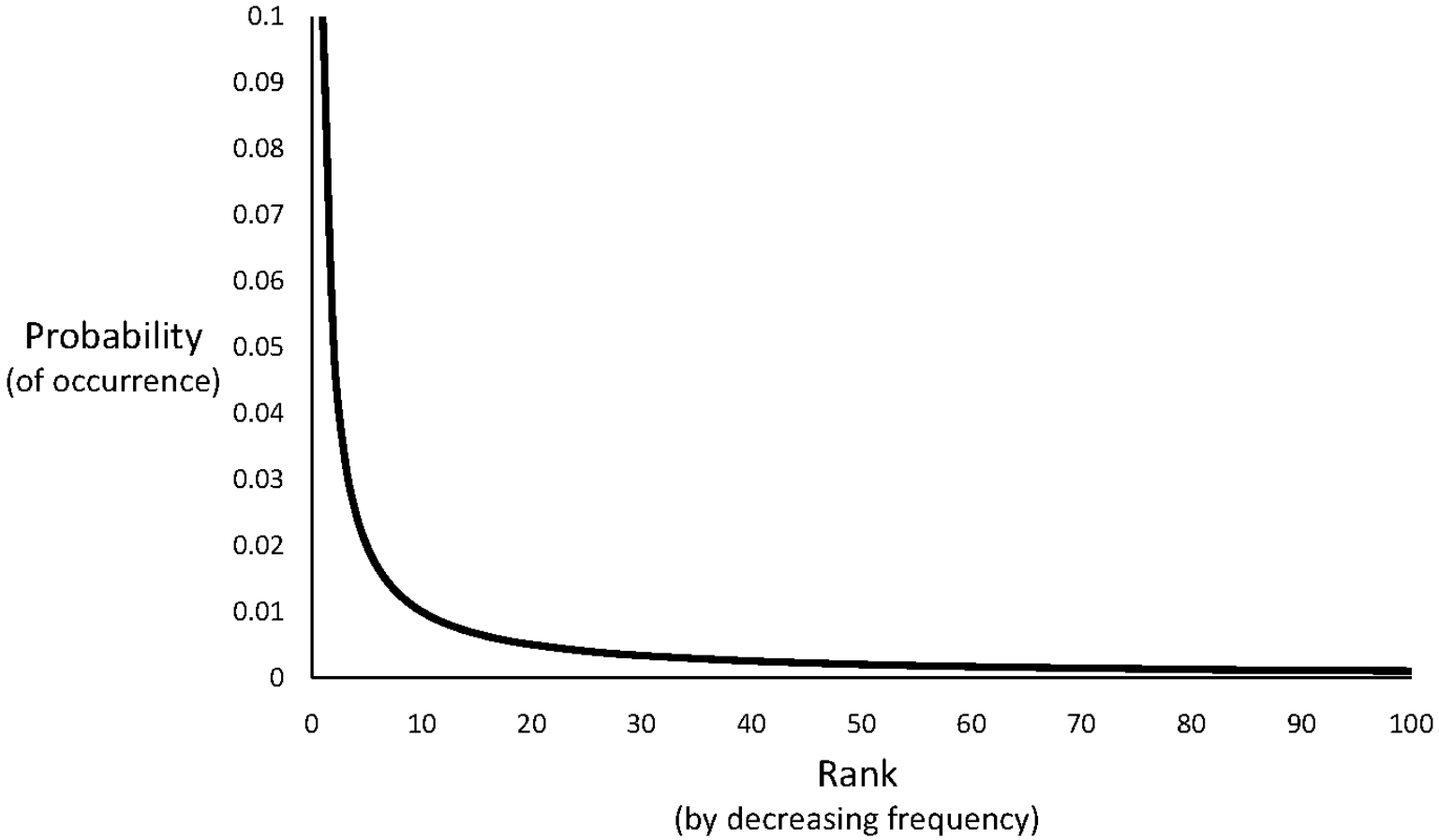
- $r \cdot p(r) = A$, where A=0.1
- $p(r) = \frac{freq(r)}{N}$
- $r \cdot p(r) = N \cdot A$

Example: Term Distribution

Word	Freq	r	Pr(%)	r*Pr
the	2,420,778	1	6.488	0.0649
of	1,045,733	2	2.803	0.0561
to	968,882	3	2.597	0.0779
a	892,429	4	2.392	0.0957
and	865,644	5	2.32	0.116
in	847,825	6	2.272	0.1363
said	504,593	7	1.352	0.0947
for	363,865	8	0.975	0.078
that	347,072	9	0.93	0.0837
was	293,027	10	0.785	0.0785
on	291,947	11	0.783	0.0861
he	250,919	12	0.673	0.0807
is	245,843	13	0.659	0.0857
with	223,846	14	0.6	0.084
at	210,064	15	0.563	0.0845
by	209,586	16	0.562	0.0899
it	195,621	17	0.524	0.0891
from	189,451	18	0.508	0.0914
as	181,714	19	0.487	0.0925
be	157,300	20	0.422	0.0843
were	153,913	21	0.413	0.0866
an	152,576	22	0.409	0.09
have	149,749	23	0.401	0.0923
his	142,285	24	0.381	0.0915
but	140,880	25	0.378	0.0944

Word	Freq	r	Pr(%)	r*Pr
has	136,007	26	0.365	0.0948
are	130,322	27	0.349	0.0943
not	127,493	28	0.342	0.0957
who	116,364	29	0.312	0.0904
they	111,024	30	0.298	0.0893
its	111,021	31	0.298	0.0922
had	103,943	32	0.279	0.0892
will	102,949	33	0.276	0.0911
would	99,503	34	0.267	0.0907
about	92,983	35	0.249	0.0872
i	92,005	36	0.247	0.0888
been	88,786	37	0.238	0.0881
this	87,286	38	0.234	0.0889
their	84,638	39	0.227	0.0885
new	83,449	40	0.224	0.0895
or	81,796	41	0.219	0.0899
which	80,385	42	0.215	0.0905
we	80,245	43	0.215	0.0925
more	76,388	44	0.205	0.0901
after	75,165	45	0.201	0.0907
us	72,045	46	0.193	0.0888
percent	71,956	47	0.193	0.0906
up	71,082	48	0.191	0.0915
one	70,266	49	0.188	0.0923
people	68,988	50	0.185	0.0925

Zipf's Law


0.1
0.09
0.08
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0
Probability
(of occurrence)
0
10
20
30
40
50
60
70
80
90
100
Rank
(by decreasing frequency)

Distribution of words in docs

- Simple model
  - considering that each word appears the same number of times in every document
- Negative binomial docs containing
  - A fraction of docs containing a word k times

$$F(k) = \binom{\alpha + k - 1}{k} p^k (1+p)^{-\alpha - k}$$

where, p, α: parameters depending on the word and docs

The number of distinct words in a doc

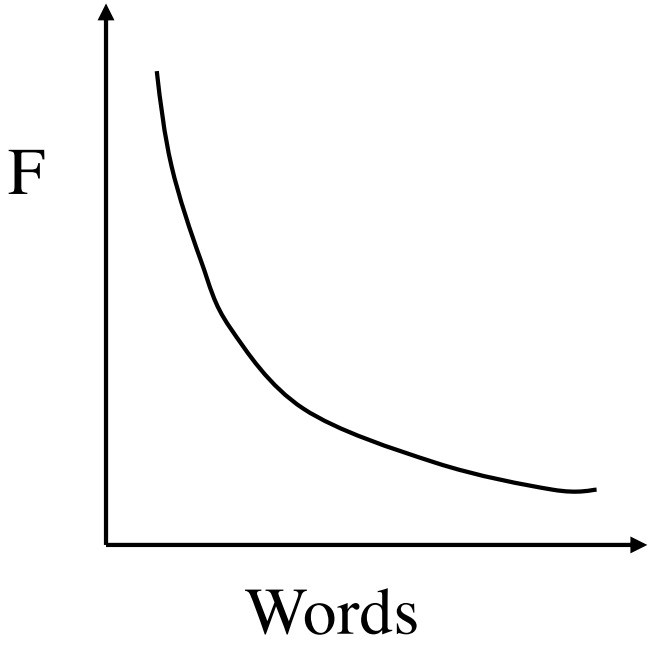
- Heaps' Law
  - predicting the **growth of the vocabulary** size in natural language text
  - vocabulary of a text of size n words:

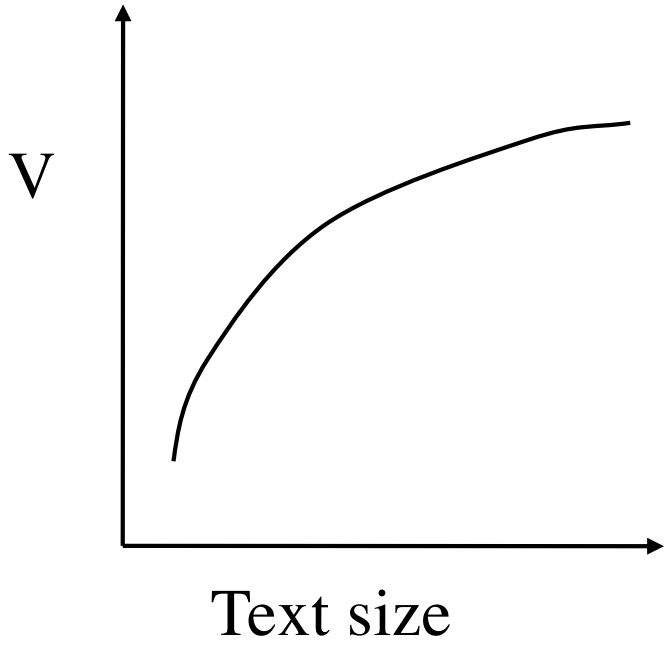
$$V = Kn^{\beta} = O(n^{\beta})$$

where, K, β: depending on the text

$10 < K < 100$, $0 < \beta < 1$

The number of distinct words in a doc





Resources

- Modern IR Chap 9
- Introduction to Information retrieval: Chap 4 & 5
- Further reading:
  - MapReduce Chap 4