IS2140 Information Storage and Retrieval

Unit 3: Index Construction and Compression

Daqing He
School of Information Sciences
University of Pittsburgh

September 17, 2018

Muddiest Points

- Bad of Word Representation
  - About the Bag of Word Representation, how to determine the relevance ranking of different words. According to the number of occurrences of the same words as those in the user's queries? How to make sure their weighted importance value? Because sometimes the degree of correlation does not depend on the count.

Muddiest Points

- Document Basics
  - Dynamic Content. In the ""basics"" section of what counts as a document, one issue not discussed is whether documents can be dynamic.
  - Is there a dynamic way to decide how to index documents based on section, chapter or book i.e. not before designing the search engine but design the search engine in such a way that it understands the user requirement and then index based on section, chapter or book?"

Muddiest Points

- N-gram
  - "In n-gram method, how do we choose the N?
  - For example, in the case of ""新西兰花"", if we choose 3 as ""n"", the answer would be related to ""新 西兰花""(fresh broccoli)or ""新西兰 花""(New Zealand flowers). Both answer I can accept, but if we choose 2 as ""n"", the result will be related to ""新西 兰花"(Xinxi Orchid, maybe Xinxi is a kind of specie), that is not acceptable."
  - Since Chinese is not suitable for segmentation indexing and one of the alternative methods introduced in the lecture is n-gram, and it produces collections of different combinations of the Chinese characters, which are different from the results using Bag of Word. Could you please introduce how to make computers understand Chinese content after doing the n-gram?"

Which n-gram?

- Three documents:
  - D1: abcd
  - D2: acde
  - D3: cdae
  - Q: abe
- Counts based on 1-gram
  - a:3, b:1, c:3, d:3, e:2
- Counts based on 2-gram
  - ab:1, bc:1, cd:2, ac:1, de:1, cd:1,da:1,ae:1
- Counts based on 3-gram
  - abc:1, bcd:1,acd:1,cde:1,cda:1, dae:1
- Counts based on 1-gram
  - D1: a:1,b:1,c:1,d:1
  - D2: a:1,c:1,d:1,e:1
  - D3: a:1,c:1,d:1,e:1
  - Q: a:1,b:1,c:1
- Counts based on 2-gram
  - D1: ab:1,bc:1,cd:1
  - D2: ac:1,cd:1,de:1
  - D3: cd:1,da:1,ae:1
  - Q: ab:1, be:1
- Counts based on 3-gram
  - D1: abc:1, bcd:1
  - D2: acd:1, cde:1
  - D3: cda:1, dae:1
  - Q: abe:1

Muddiest Points

- Stemming
  - There are two approaches to process tokens mentioned in the lecture. The first one is lemmatization, the other one is stemming. When should we use lemmatization rather than stemming?
- Phrases
  - "Since POS tagging is slower than statistical analysis, why is POS tagging still been used often in the IR community and can we use statistical analysis for phrase recognition in all cases?"

Agenda

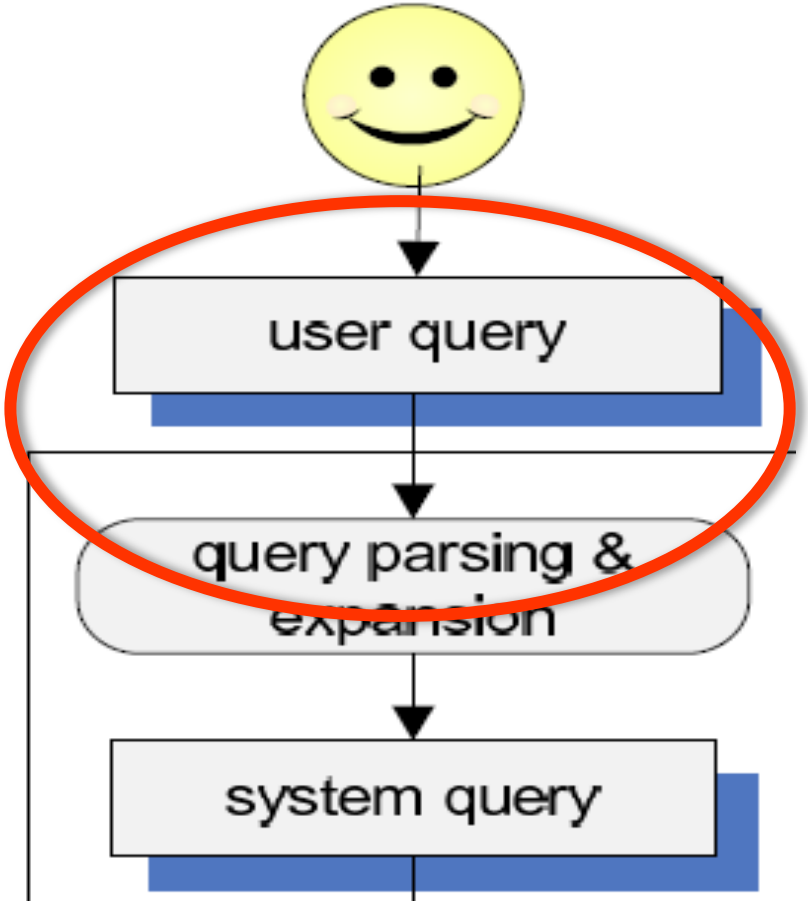
- Query Processing
- Index Construction
- Index Compression

Class Goals

- After this class, you should be able to
  - know the basic ideas of constructing index
  - Know the reasons why index is constructed into such inverted format with various components
  - Know the basics of index compression
  - Can design simple index construction and compression modules

Query Processing

Query Processing



Query Processing and Doc Indexing

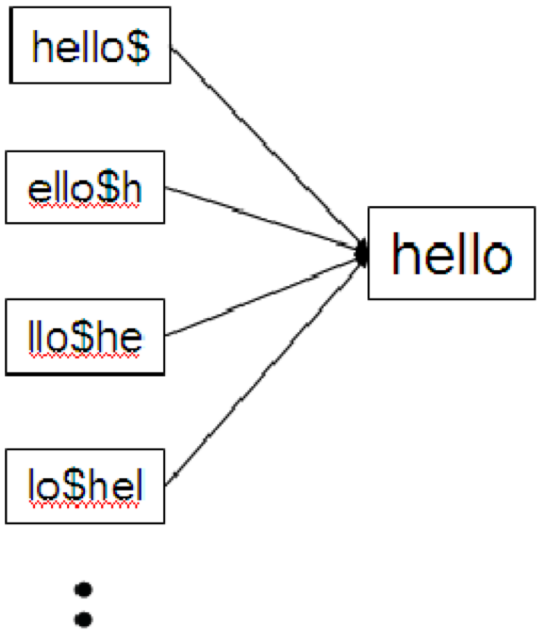
- Query processing is related to Doc Indexing
  - These two have to be comparable
  - Perform the exact steps as in indexing, for example
    - Remove stopwords if stopwords are removed in indexing
    - Stemming query terms if stemming is used in indexing
    - Phrases are identified if phrases are important in indexing

Wildcard Queries

- Stemming helps in coping with morphological variations
- But users still need flexibilities in their query input
- Reasons:
  - Users would like to control the variations too: e.g. wom*n for woman and women
  - Users do not know the right spelling: e.g. S*dney for Syndey and Sindey
  - Users is not sure whether stemming is involved or not
  - Users is not sure the right forms of foreign words Universit* Stuttgart
- Retrieval system has to provide wildcard query capabilities
  - Then how retrieval system to deal with those wildcard queries

Permuterm Indexes for Wildcard Queries

- At index part, using a special symbol $ mark end of a term
  - So hello now becomes hello$
  - Then rotate the sequence of "hello$" as shown
  - Save all different rotations in a database and all point to the original term
- When query comes, change to the same format
  - Say query term is h*llo, then it changes to
    h*llo$ => llo$h*
  - So look for all the strings match to "llo$h"
    - Therefore, return hallo, hello, hullo



N-gram for Wildcard Queries

- At index part, divide terms into strings with n characters, say trigram
  - castle => $ca, cas, ast, stl, tle, le$
  - Save all different strings in a separate index between n-gram patterns and their corresponding original terms
    - $ca → cable, caboodle, cackle, cagoule, cajole, calculable ….
- When query term comes, change to the same format too
  - Say query term is ca*le, then it changes to

    ca*le => ca, le
  - So look for all the strings match to "$ca" and "le$"
    - Therefore, return cable, caboodle, cackle, cagoule, cajole, calculable, camisole, camomile, campanile, candle, capable, capsule, castle, cattle

Spelling Check

- Is a query term misspelled? If so, make correction suggestion
- Check misspell
  - Based on dictionary (not in the dictionary)
  - Based on search results (returned docs are lower than certain number)
- Make correction suggestions
  - Resources
    - Based on dictionary
    - Based index terms in the collection
    - Based on query logs
  - Methods
    - Edited distance
    - N-gram overlapping (see IIR book)

Query Languages

- Every full text search system has an underlying query language
- Two types of languages
  - Boolean or structured,
  - Free text or bag-of-words queries
- Many systems support a combination of both
- Even simple bag-of-words queries have underlying structure
  - Google assumes default operator "AND"
  - Some other systems take "OR"

Example of Query Language: Indri - I

- handle both simple keyword queries and extremely complex queries.
  - allows complex phrase matching, synonyms, weighted expressions, Boolean filtering, numeric (and dated) fields, and the extensive use of document structure (fields), among others.

```
query   :=  ( beliefOp )+

beliefOp  :=  "#weight" ( extentRestrict )? weightedList
    | "#combine" ( extentRestrict )? unweightedList
    | "#or" ( extentRestrict )? unweightedList
    | "#not" ( extentRestrict )? '(' beliefOp ')'
    | "#wand" ( extentRestrict )? weightedList
    | "#wsum" ( extentRestrict )? weightedList
    | "#max" ( extentRestrict )? unweightedList
    | "#prior" '(' FIELD ')'
    | "#scoreifnot | #filrej" '(' unscoredTerm beliefOp ')'
    | "#scoreif | #filreq" '(' unscoredTerm beliefOp ')'
    | termOp ( '.' fieldList )? ( '.' '(' fieldList ')' )?
```

Example of Query Language: Indri - II

- Core concepts in Indri Query language
  - Term: a term in the index (e.g., "white")
  - Extent: a span within a document (e.g., Body, Title)
    - Useful for documents with fields
  - Weight: a positive number
  - Term Operator: generates a new index term dynamically
    - Allow some representation of synonyms #syn (astronaut, cosmonaut)
  - Belief operator: an operator that combines scores
    - E.g. #combine(george bush)
    - #weight(1.0 george 3.0 bush)
    - #weight (2.0 #or(president george) 3.0 bush)
    - #combine is a #weight operator where all weights are equal

Example of Query Language: Indri - III

- Query with Extents
  - Extents are fields and annotations in documents
  - Fields: e.g., <TITLE> … </TITLE>
  - Annotations: e.g. <ENAMETYPE=PERSON>Bush</ENAMEX>
  - Example queries
    - #combine[sentence](napolean elba)
    - #combine[passage 100:50](napolean elba)
      - Retrieve 100-word passages, score using 50-word windows
    - #combine(#1(elvis died on #any:DATE))
      - #n requires that terms occur in order, spearated by < n term
      - #1(bill clinton) matches to "bill clinton" but not "bill j clinton"
      - #any matches any term

Example of Query Language: Indri - IV

- Indri Query language has many other features
- Consult Indri query language site at http://www.lemurproject.org/lemur/IndriQueryLanguage.php

Query Processing Steps

- "stop-phrase" removal
  - E.g. "I would like to find documents about …"
- Stopword removal and stemming if they are done in indexing
- Recognize noun phrases
  - Using hyphenation: e.g., short-term => #1(short term)
  - Using POS tagging: e.g., white house => #1 (white house)
  - Using capitalization: e.g., Daqing He => #1(Daqing He), #1(Daqing He).Person
- Weight query terms by frequency
  - E.g., George Bush, Bush White House => 1 George 2 Bush 1 White 1 House

Queries from TREC Topics - I

<topic>

<number>HARD-402</number>

<title>Identity Theft</title>

<description>How safe is your information on the Internet? </description>

<topic-narrative>

I am looking for an examination of the rising awareness and occurrence of identity theft. I would like to see information about how people can protect themselves, various ways people may have their identities stolen, and measures one can take to prevent identity theft. Is information safe on the Internet? I am looking specifically for Internet identity theft.

</topic-narrative>

</topic>

Resemble a user issued query

Resemble a one sentence question

Resemble a need statement

Queries from TREC Topics - II

- Title only query, e.g. (not necessarily the best representation)

#combine(#PHRASE (identity theft) identity theft)

- Query based on title and description fields

#wsum(1 4.0 #UW3(identity theft) 4.0 identity 4.0 theft
2.0 safe 2.0 information 2.0 Internet)

Index

A Search Scenario

- Information Need: Identity Theft Protection Internet

- Document Set:
  - Doc 1: Apple introduces iphone via internet, iphone can access internet
  - Doc 2: police uses internet to look for the identity of an armed killer. His photos are all over the internet.
  - Doc 3: about 3 million cases of identity theft each year in US, most of them happened on the Internet. Consumers need more protections of their identities
  - Doc 4: Credit card companies lose millions due to identity frauds on the Internet. Each company has millions of consumers

A Match between Query and Docs

- Boolean Query: Identity AND Theft AND (Protection OR Internet)
  - Processed query: ident AND theft AND (protect OR internet)
- Document Representation: (after tokenization, normalization, remove stopwords, and stemming)
  - Doc 1: appl, introduc, iphon, internet, iphon, access, internet
  - Doc 2: polic, use, internet, look, ident, arm, killer, photo, internet
  - Doc 3: 3, million, case, ident, theft, year, us, happen, internet, consum, need, protect, ident
  - Doc 4: credit, card, compani, lose, million, ident, fraud, internet, compani, million, consum

Problems with This Approach

- Very inefficient.
  - Access docs multiple times to retrieve complete information about a query term
    - Problem: all information about a query term is not put together
    - Problem: no carefully designed structure to hold the information
  - Need many unnecessary comparisons to find the query terms in docs
    - Problem: do not know where the query terms are
    - Problem: do not know which doc contain query terms
- Very ineffective
  - Hard to know the importance of the terms in docs
    - Problems: do not know whether the term appear in a doc
    - Problems: do not know how many docs contain a term

Index Construction

So Features of Index - I

- requirement: given a term, all related information needs to be easily accessible
  - Solution: create a vocabulary list, a dictionary like structure

Terms
…
compani
consum
…
ident
internet
…
protect
theft
…
year

So Features of Index - II

- requirement: which documents contain a terms need to be available
  - Solution: an array telling which docs contain the terms

Terms	Doc 1	Doc 2	Doc 3	Doc 4
…				
compani	0	0	0	1
consum	0	0	1	1
…				
ident	0	1	1	1
internet	1	1	1	1
…				
protect	0	0	1	0
theft	0	0	1	0
…				
year	0	0	1	0

Query:

ident AND theft AND (protect OR internet)

So works, but can you see any problem?

Problems

- Observations:
  - Still not quick to find which documents contain a term
  - The term-document matrix is very sparse
  - no information about some terms are more useful than others
- Can we make this data structure even more efficient for fast retrieval?

So Features of Index - III

- Postings: avoid data sparse and explicitly indicate docs
  - Solution: an array of only the docs contain the terms

Terms	Doc 1	Doc 2	Doc 3	Doc 4	Postings
…					
compani	0	0	0	1	4
consum	0	0	1	1	3,4
…					
ident	0	1	1	1	2,3,4
internet	1	1	1	1	1,2,3,4
…					
protect	0	0	1	0	3
theft	0	0	1	0	3
…					
year	0	0	1	0	3

A Simple Inverted Index

Terms	Postings
…	
compani	4
consum	3,4
…	
ident	2,3,4
internet	1,2,3,4
…	
protect	3
theft	3
…	
year	3

Still can you see any problem?

Problems

- Observations:
  - no position information for proximate queries
  - no information about some terms are more useful than others

A more Elaborated Inverted Index

Terms	Postings
…	
compani	(4: 3, 14)
consum	(3 :19),(4 :18)
…	
ident	(2:8), (3: 6,25), (4:8)
internet	(1:5,9), (2:3,19), (3:18), (4:12)
…	
protect	(3:22)
theft	(3:7)
…	
year	(3:9)

Query:

uw3(ident theft)

Simple Exercise

- A collection of two documents
  - Doc1: Wrecked Roads and Bridges in Chile Hinder Rescue Effort in Chile Earthquake
  - Doc2: Test for Chile's Coalition in Presidential Election in Chile
- After stopword removing and stemming
  - Doc1: wreck road bridg chile hinder rescu effort chile earthquak
  - Doc2: test chile coalit presidenti elect chile

- What is the vocabulary? What is the inverted file looks like if we want to record how many times a term appears in a document?

A Solution

Terms	Postings
bridg	(1#1)
chile	(1#2), (2#2)
coalit	(2#1)
earthquak	(1#1)
effort	(1#1)
elect	(2#1)
hinder	(1#1)
presidenti	(2#1)
rescu	(1#1)
road	(1#1)
test	(2#1)
wrect	(1#1)

If want to record the position of "bridg" in Doc 1, is it 3rd words Or 4th words?

So how big the index would be?

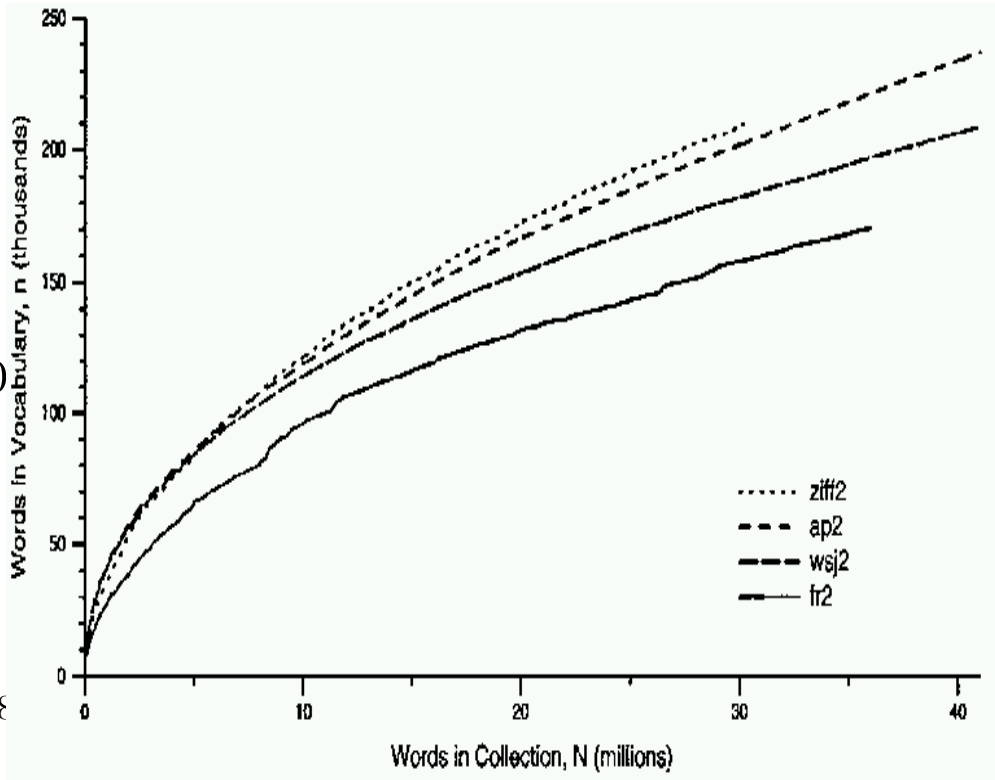
How big is the vocabulary?

- Heap's Law tells us about vocabulary size

$$V = Kn^{\beta}$$

V is vocabulary size
n is corpus size (number of words)
K and β are constants

  - Depends on text, 10<= K <= 100
  - In English, 0.4 <= β <= 0.6
    - approx. square-root
  - when $K \approx 20, \beta \approx 0.5$
    - 100,000 words, voc size = 6325
    - 1,000,000 words, voc size = 20000



What could be the new words?

- When adding new documents, the system is likely to have seen most terms already

- But vocabulary does not really upper-bounded with the increase of the size of the collection

Zipf's Law

- Zipf's law related a term's frequency to its rank
  - Rank the terms in a vocabulary by frequency, in descending order

$$R\, f_R = A\, N$$

  - f_R : frequency of term ranked R
  - N: total number of words occurred
  - Empirical observation, for English A = 0.1

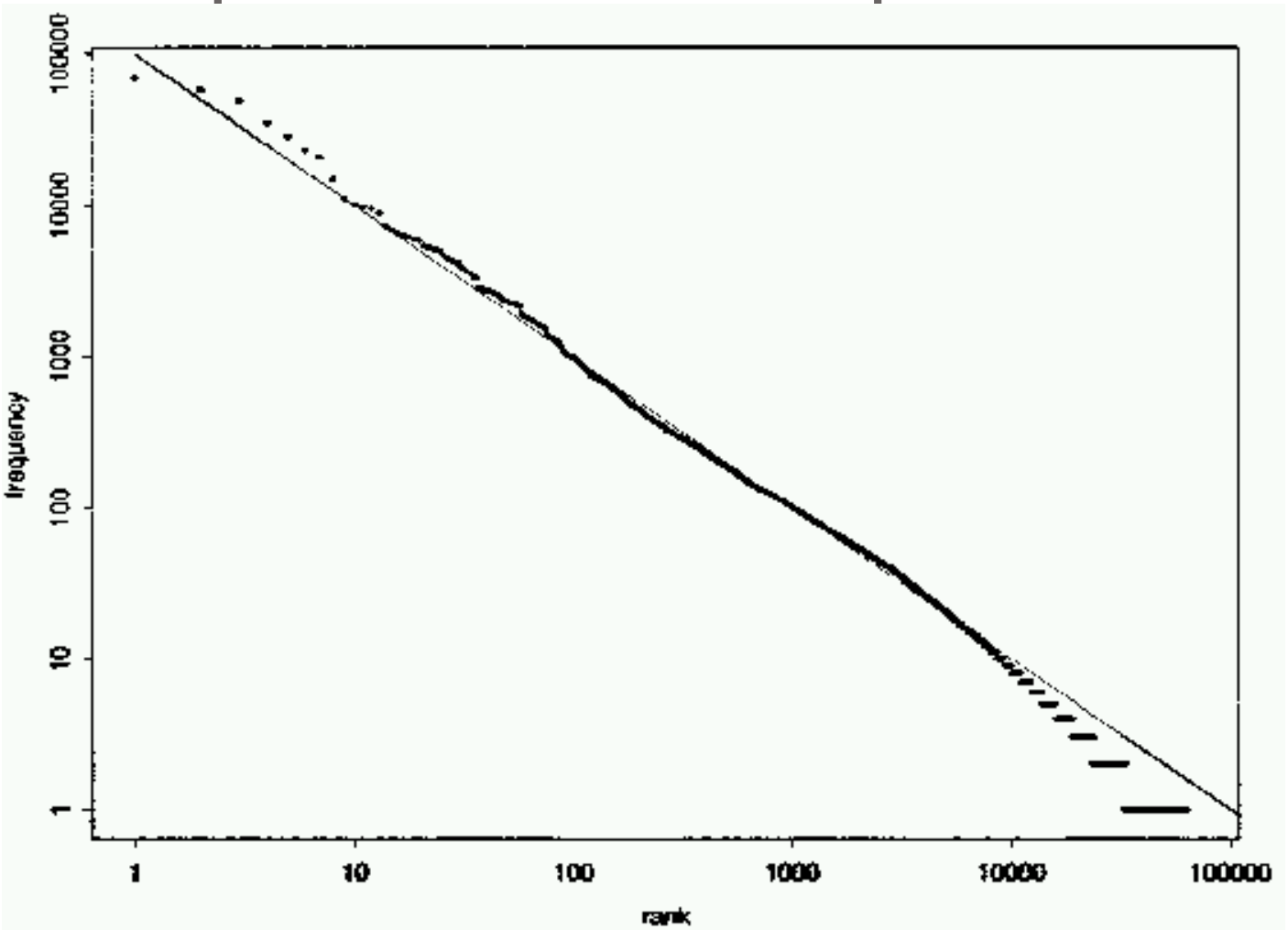
Does Real Data Fit Zipf's Law?

- A law of the form $y = kx^c$ is called a power law.
- Zipf's law is a power law with $c = -1$
- On a log-log plot, power laws give a straight line with slope c.

- Zipf is quite accurate except for very high and low rank.

$$\log(y) = \log(kx^c) = \log k + c\log(x)$$

Fit to Zipf for Brown Corpus



$$k = 100{,}000$$

Zipf on WSJ '87-92

Rank	Term	Zipf	Actual	Rank	Term	Zipf	Actual
1	the	7,831,076	4,352,160	101	9	77,535	80,490
2	of	3,915,538	2,134,125	102	most	76,775	80,409
3	to	2,610,358	2,023,402	103	such	76,030	80,037
4	a	1,957,769	1,811,373	104	time	75,299	80,014
5	in	1,566,215	1,546,782	105	no	74,582	78,459
6	and	1,305,179	1,507,140	106	into	73,878	78,208
7	s	1118725	855,190	107	only	73,188	78,150
8	that	978,885	787,792	108	trading	72510	78,133
9	for	870,120	780,138	109	many	71,845	77,578
10	is	783,107	605,988	110	so	71,192	77,099
11	said	711,915	528,481	111	now	70,550	76,281
12	it	652,590	510,102	112	based	69,920	75,798
13	on	602,390	483,160	113	prices	69,302	73,536
:	: :	: : : :	: : : :	:	: :	: : :	: : :

Sample Word Frequency Data

(from B. Croft, UMass)

Frequent Word	Number of Occurrences	Percentage of Total
the	7,398,934	5.9
of	3,893,790	3.1
to	3,364,653	2.7
and	3,320,687	2.6
in	2,311,785	1.8
is	1,559,147	1.2
for	1,313,561	1.0
The	1,144,860	0.9
that	1,066,503	0.8
said	1,027,713	0.8

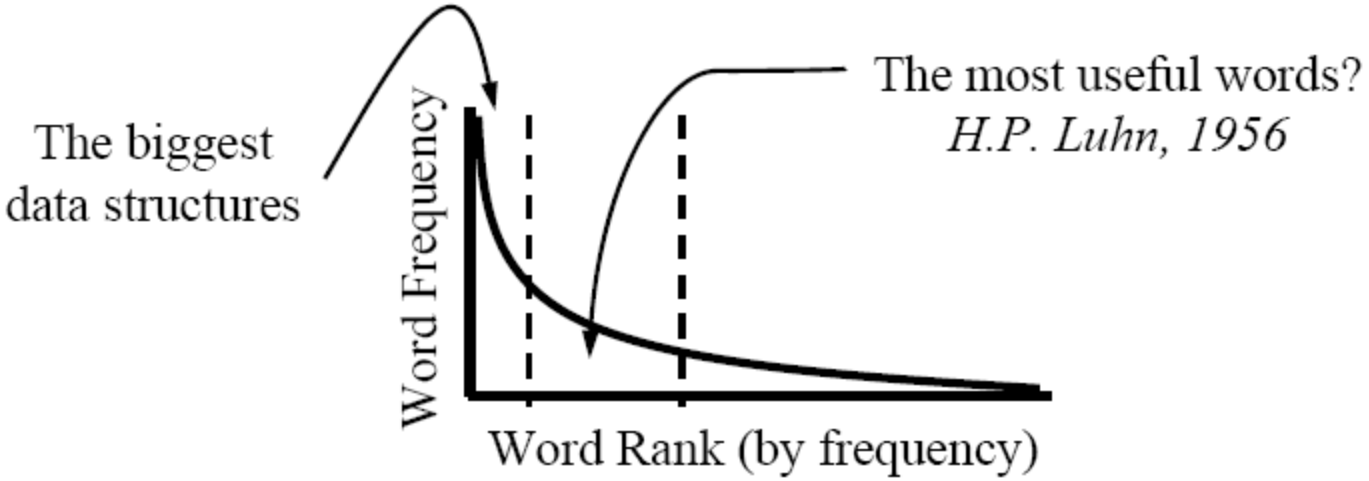
Frequencies from 336,310 documents in the 1GB TREC Volume 3 Corpus
125,720,891 total word occurrences; 508,209 unique words

Zipf's Law Impact on IR

- Good News: Stopwords will account for a large fraction of text so eliminating them greatly reduces inverted-index storage costs.
  - take up 50% of the text.
- Bad News: For most words, gathering sufficient data for meaningful statistical analysis (e.g. for correlation analysis for query expansion) is difficult since they are extremely rare.

Statistical Properties of Text

- **Summary:**
  - Term usage is highly skewed, but in a *locally predictable* pattern
- **Why it is important to know the characteristics of text**
  - optimization of data structures
  - statistical retrieval algorithms depend on them



How big are the postings?

- Very compact when only consider which term in which doc
  - About 10% of the size of the documents
- Not much larger when also consider frequency of a term in a doc
  - Perhaps 20% of collection size
- Enormous for proximity operators when position is also needed
  - Sometimes larger than the document collection

Terms	Postings
…	
compani	4
consum	3,4
…	
ident	2,3,4
internet	1,2,3,4
…	
protect	3
theft	3
…	
year	3

Terms	Postings
…	
compani	(4: 3, 14)
consum	(3 :19),(4 :18)
…	
ident	(2:8), (3: 6,25), (4:8)
internet	(1:5,9), (2:3,19), (3:18), (4:12)
…	
protect	(3:22)
theft	(3:7)
…	
year	(3:9)

Effects of Doc Processing to Index Size

	(distinct) terms			non-positional postings		
	number	Δ%	T%	number	Δ%	T%
unfiltered	484,494			109,971,179		
no numbers	473,723	−2	−2	100,680,242	−8	−8
case folding	391,523	−17	−19	96,969,056	−3	−12
30 stop words	391,493	−0	−19	83,390,443	−14	−24
150 stop words	391,373	−0	−19	67,001,847	−30	−39
stemming	322,383	−17	−33	63,812,300	−4	−42

▶ **Table 5.1** The effect of preprocessing on the number of terms, non-positional postings, and positional postings for RCV1. "Δ%" indicates the reduction in size from the previous line, except that "30 stop words" and "150 stop words" both use "case folding" as their reference line. "T%" is the cumulative reduction from unfiltered. We performed stemming with the Porter stemmer (Chapter 2, page 33).

Hardware Issues

- Storage Hierarchy
  - Size: External storage > Hard drive > Main memory > CPU Caches
    - Unlimited space to only few hundred MBytes
  - Speed: External storage < Hard drive < Main memory < CPU Caches
    - Very slow to as fast as CPU
  - Implications to the storage of index?
- Hard drive has seek time where the disk heads need to move to the part of the disk
  - No data are transferred in this time
  - Implications to the storage of index?

Why is size important?

- RAM
  - Typical size: 4 to 8 GB
  - Typical access speed: 50 ns
- Hard drive:
  - Typical size: 500 to 1000 GB (my laptop)
  - Typical access speed: 10 ms = 10,000,000 ns
- Hard drive is 200,000x slower than RAM!

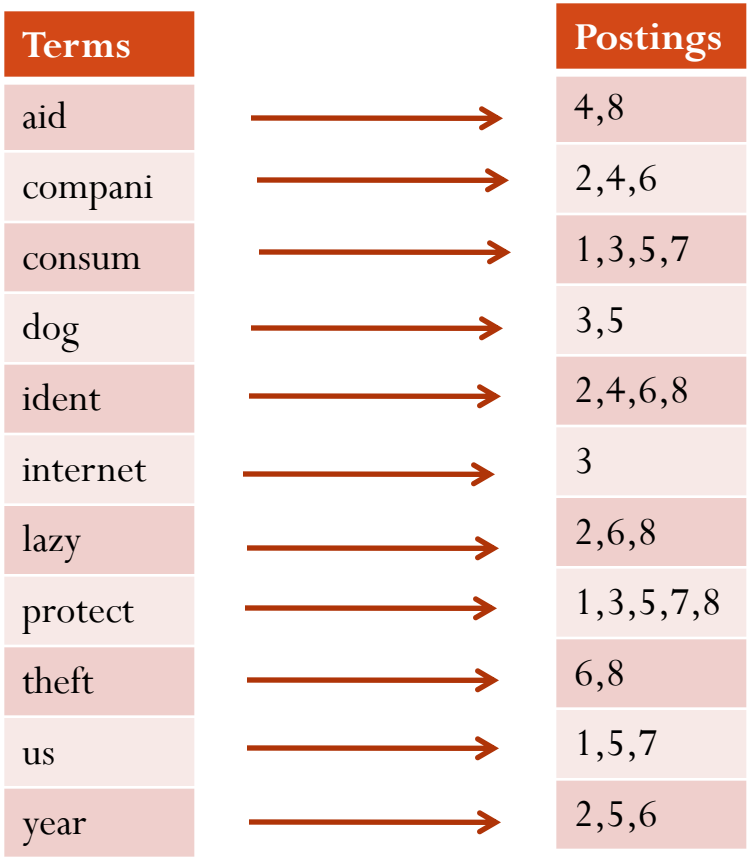
Hardware Issues - II

- Operating system often read/write entire blocks of data
  - The time to read/write a byte is equal to that of the whole block
    - Block size can be 8, 16, 32, and 64 KBytes
  - Implications to the storage of index?
- Data transfer between disk and memory is handled by bus,
  - so the CPU is available for use during the time
  - Implications to the storage of index?

Some practical issues

- Size of Index could be too big to be stored in RAM
  - Too slow if stored on hard drive
  - Need way to keep important and relevant index part in RAM
  - Others are on hard drive
- Operations needed to look for a term in term list can be huge
  - better data structure

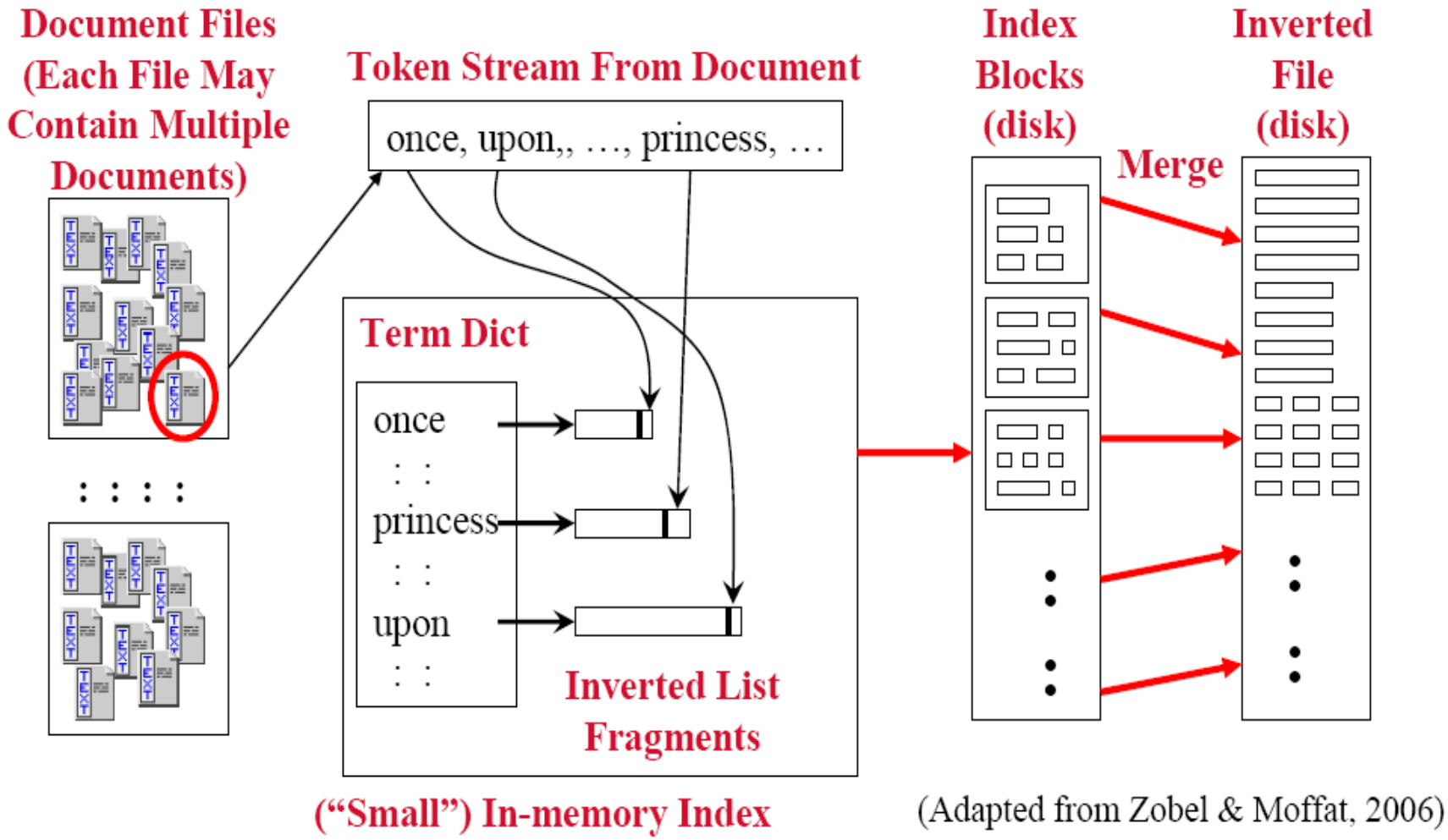
Decoupling the Inverted Index



Basic components of Index

- Term dictionary
  - The vocabulary of the index terms
- Posting lists
  - Info about where each term occurs in the corpus
- Document vector
  - Info about every term in a document
- Field mask (maybe)
  - Info about each field in a document
- And many other info

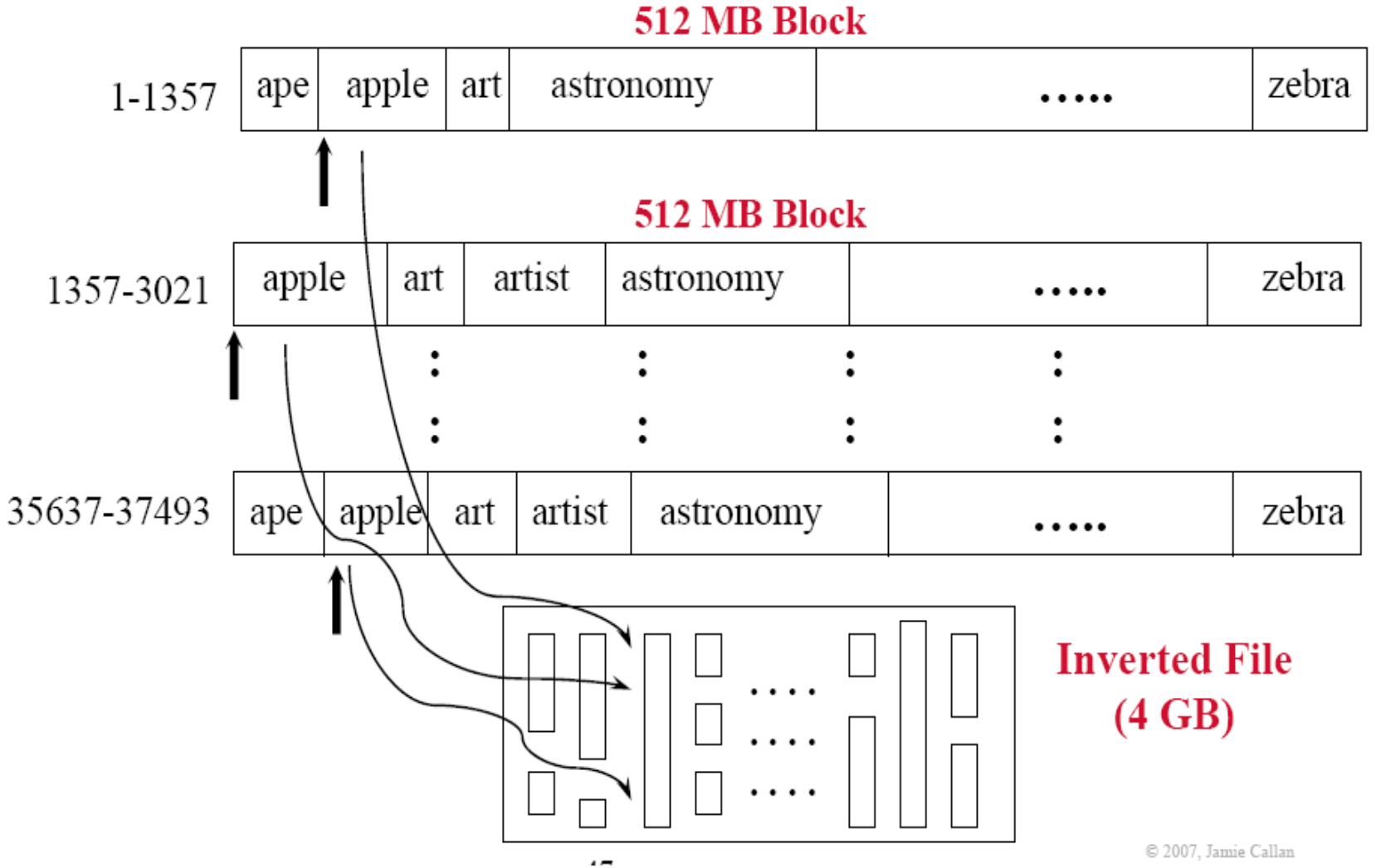
How Inverted Files are Built



How Inverted Files are Built

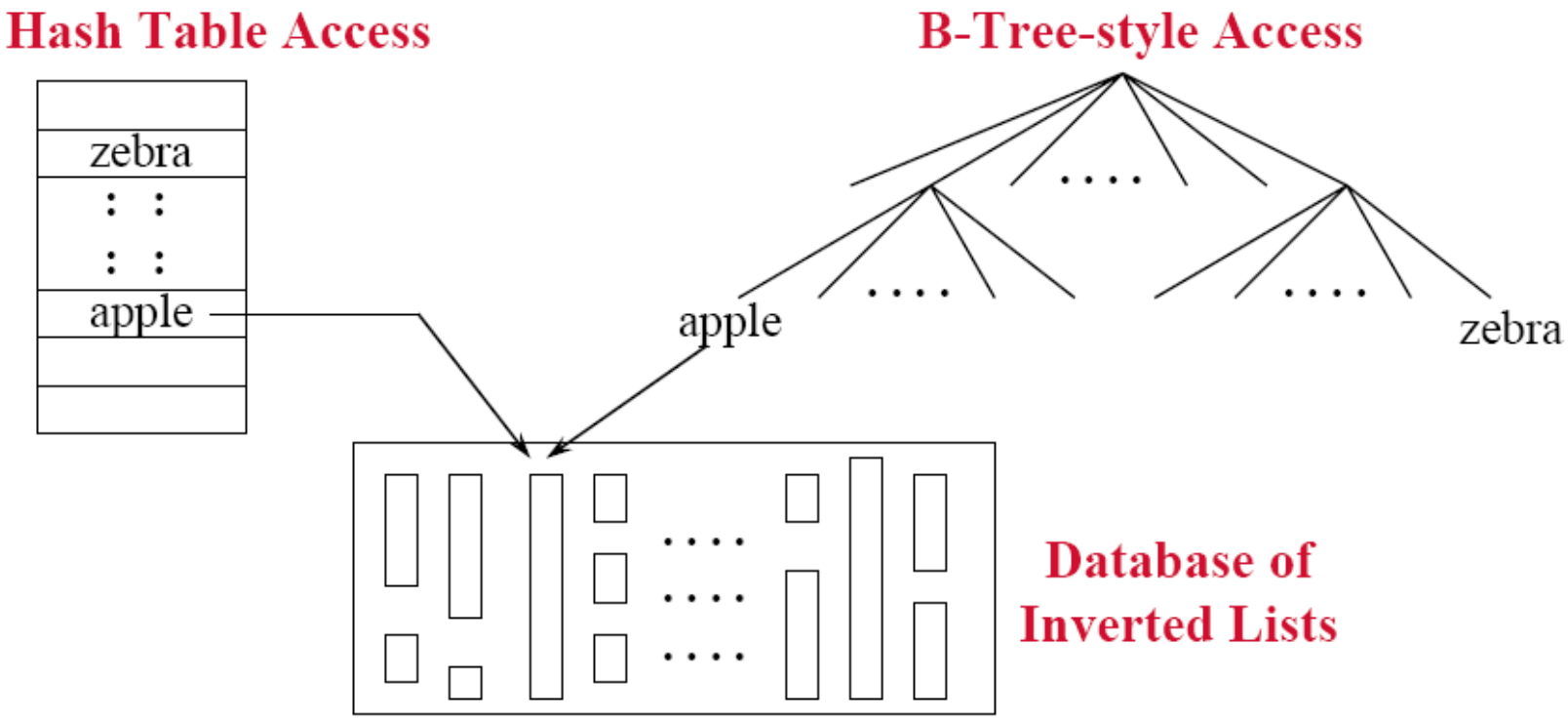
- The in-memory index stores
  - A portion of the term dictionary, fragments of posting lists
- The in-memory index is small compared to the full index
  - Perhaps 1-5% the size of total
- When in-memory index buffers are full
  - Flush to disk
  - Reinitialize in-memory index
  - Continue parsing
- When all document are parsed, all blocks of index are merged
  - Very fast – essentially a merge sort

How Inverted Files are Built : Merging Index Blocks



Access Methods to Inverted File

- Enable accurate and efficient document retrieval
- Organize the term list with efficient access methods



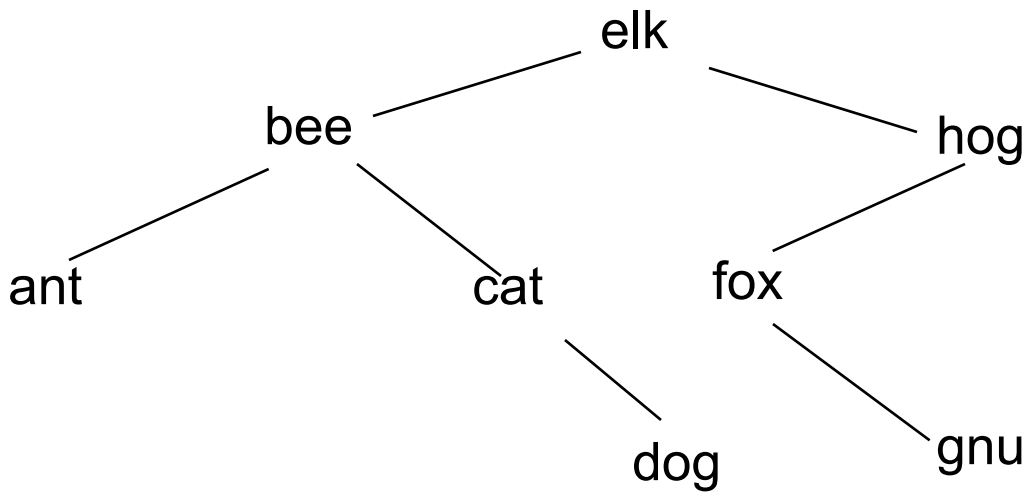
Data Structures for Access Method

- Hash Table
  - Very fast, O(1) look up, but could suffer if the terms not distributed evenly
  - Usually a big in-memory data structure
  - Could be complex to expand
- B-Tree
  - Fast, O(log n) lookup, but terms guaranteed to distribute evenly
  - Often implemented as a two-stage data structure
    - Frequent terms in memory, infrequent terms on disk
    - Efficient use of RAM, occasional slow access

Binary Tree

- A tree structure where
  - Each node has at most two children
  - The left child comes before the parent
  - The right child comes after the parent

Input: elk, hog, bee, fox, cat, gnu, ant, dog



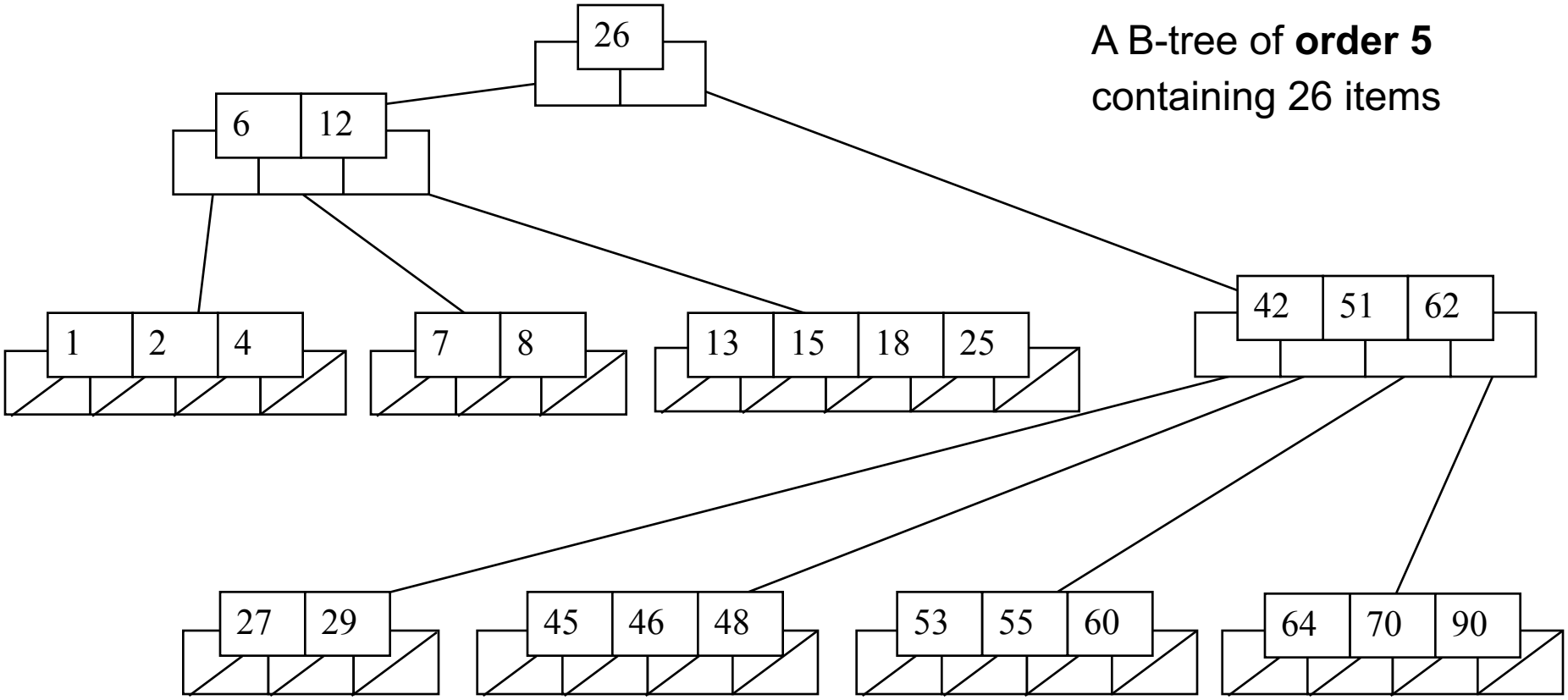
Binary Trees

- Advantages
  - Can be searched quickly
  - Easy to add an extra term
  - Economical use of storage (although less so than linear term list)
- Disadvantages
  - The shape of a binary trees depends on insertion order of terms
    - What's the worst case scenario?
  - Trees tend to become unbalanced
    - So need to balance trees

Definition of a B-tree

- A B-tree of order m is an m-way tree (i.e., a tree where each node may have up to m children) in which:
  1. the number of keys in each non-leaf node is one less than the number of its children and these keys partition the keys in the children in the fashion of a search tree
  2. all leaves are on the same level
  3. all non-leaf nodes except the root have at least $\lceil m / 2 \rceil$ children
  4. the root is either a leaf node, or it has from two to m children
  5. a leaf node contains no more than $m - 1$ keys
- The number m should always be odd

An example B-Tree



A B-tree of **order 5** containing 26 items

Note that all the leaves are at the same level

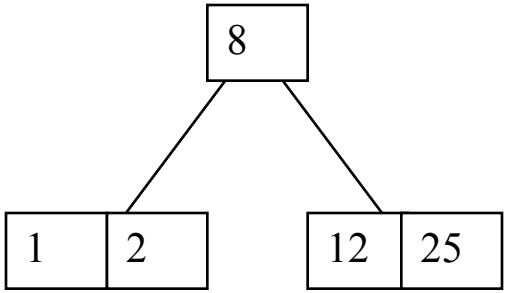
Constructing a B-tree

- Suppose we start with an empty B-tree and keys arrive in the following order:1 12 8 2 25 5 14 28 17 7 52 16 48 68 3 26 29 53 55 45
- We want to construct a B-tree of order 5
- The first four items go into the root:

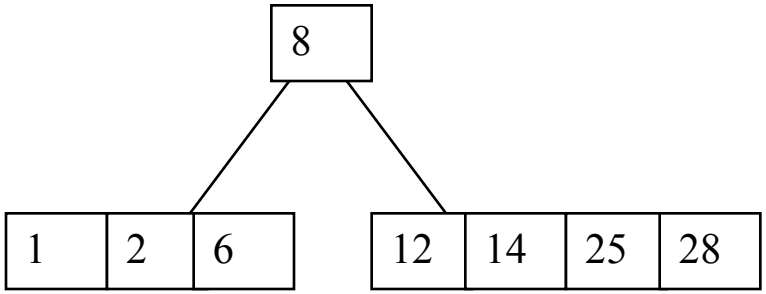
1	2	8	12

- To put the fifth item in the root would violate condition 5
- Therefore, when 25 arrives, pick the middle key to make a new root

Constructing a B-tree (contd.)

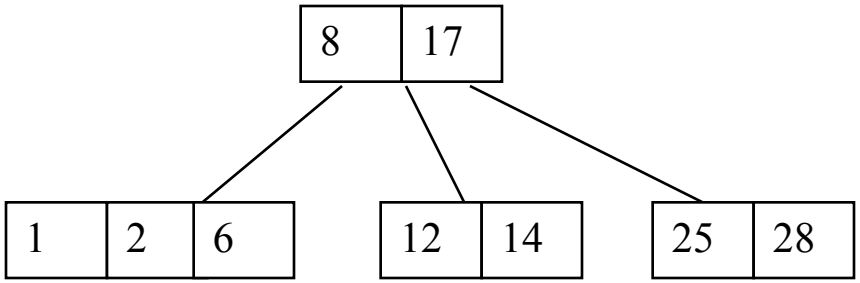


6, 14, 28 get added to the leaf nodes:

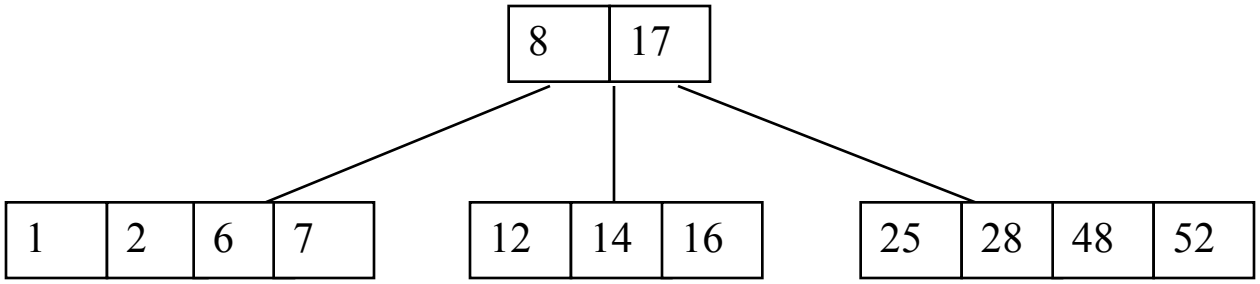


Constructing a B-tree (contd.)

Adding 17 to the right leaf node would over-fill it, so we take the middle key, promote it (to the root) and split the leaf

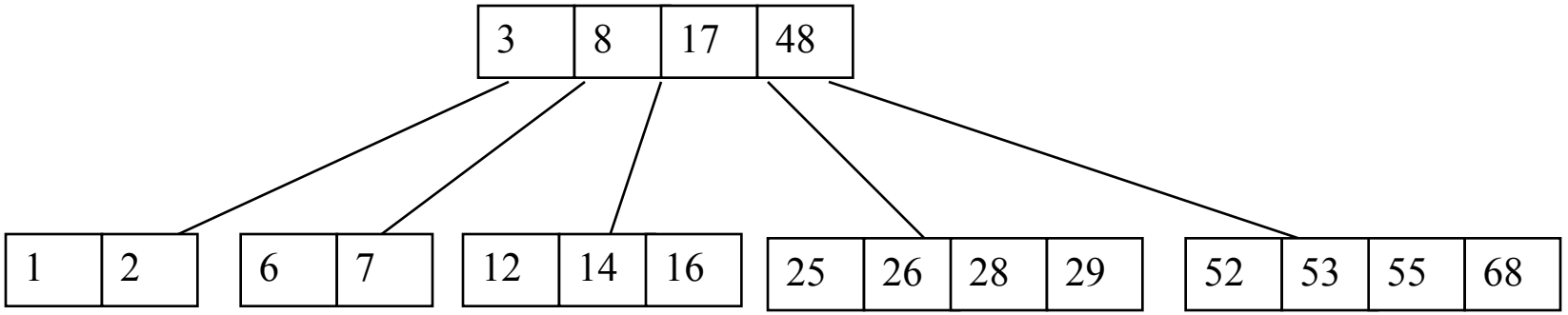


7, 52, 16, 48 get added to the leaf nodes

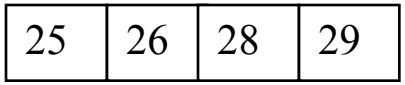


Constructing a B-tree (contd.)

Adding 68 causes us to split the right most leaf, promoting 48 to the root, and adding 3 causes us to split the left most leaf, promoting 3 to the root; 26, 29, 53, 55 then go into the leaves

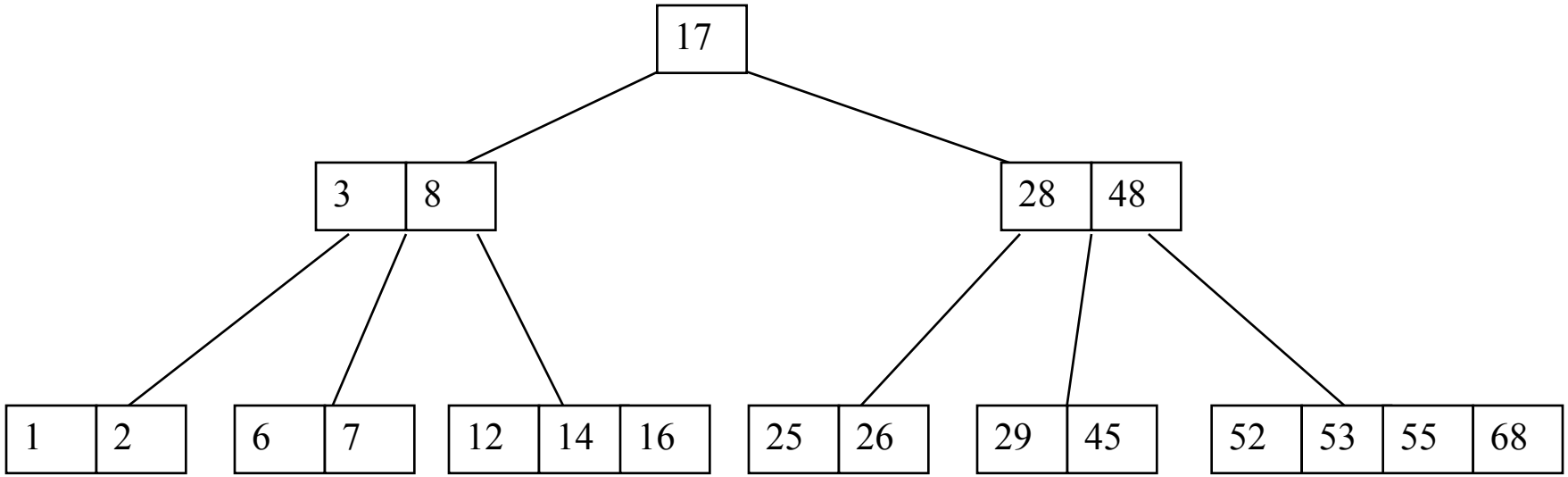


Adding 45 causes a split of



and promoting 28 to the root then causes the root to split

Constructing a B-tree (contd.)



Index Compression

Index Compression

- Problems:
  - The size is still a problem
- Opportunities
  - Compress dictionary terms
  - Compress posting lists

Why Compress Dictionary Terms

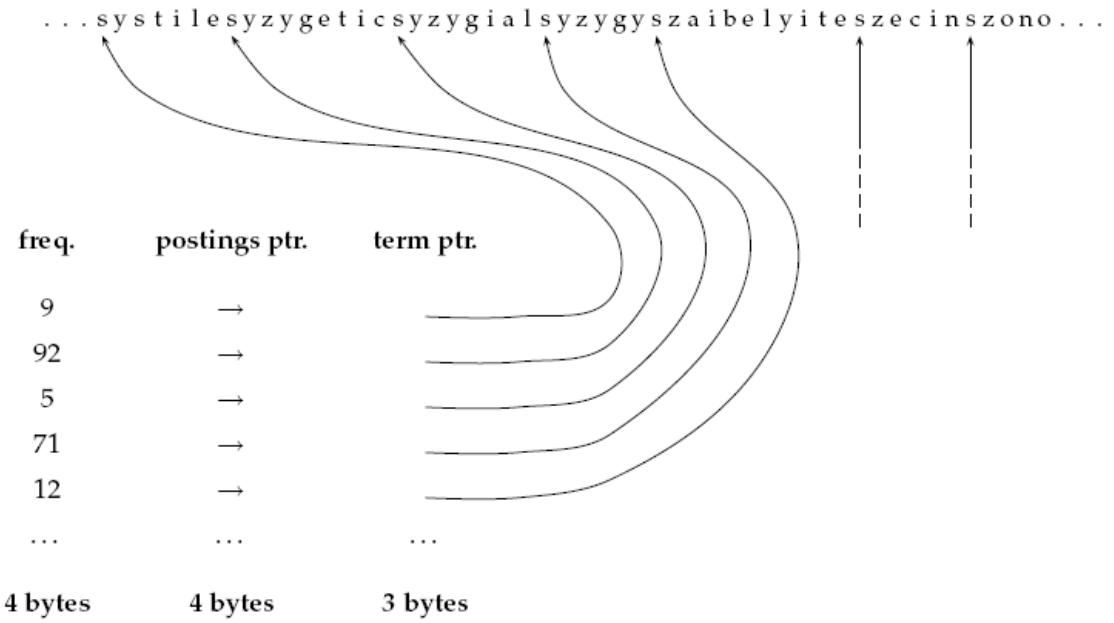
term	freq.	pointer to postings list
a	656,265	⟶
aachen	65	⟶
…	…	…
zulu	221	⟶
space needed:	20 bytes	4 bytes	4 bytes

► **Figure 5.3** Storing the dictionary as an array of fixed-width entries.

- With 4 bytes for the pointer postings, how big the vocabulary can be
- If we have a collection with 400,000 unique terms, how big the dictionary term list is?

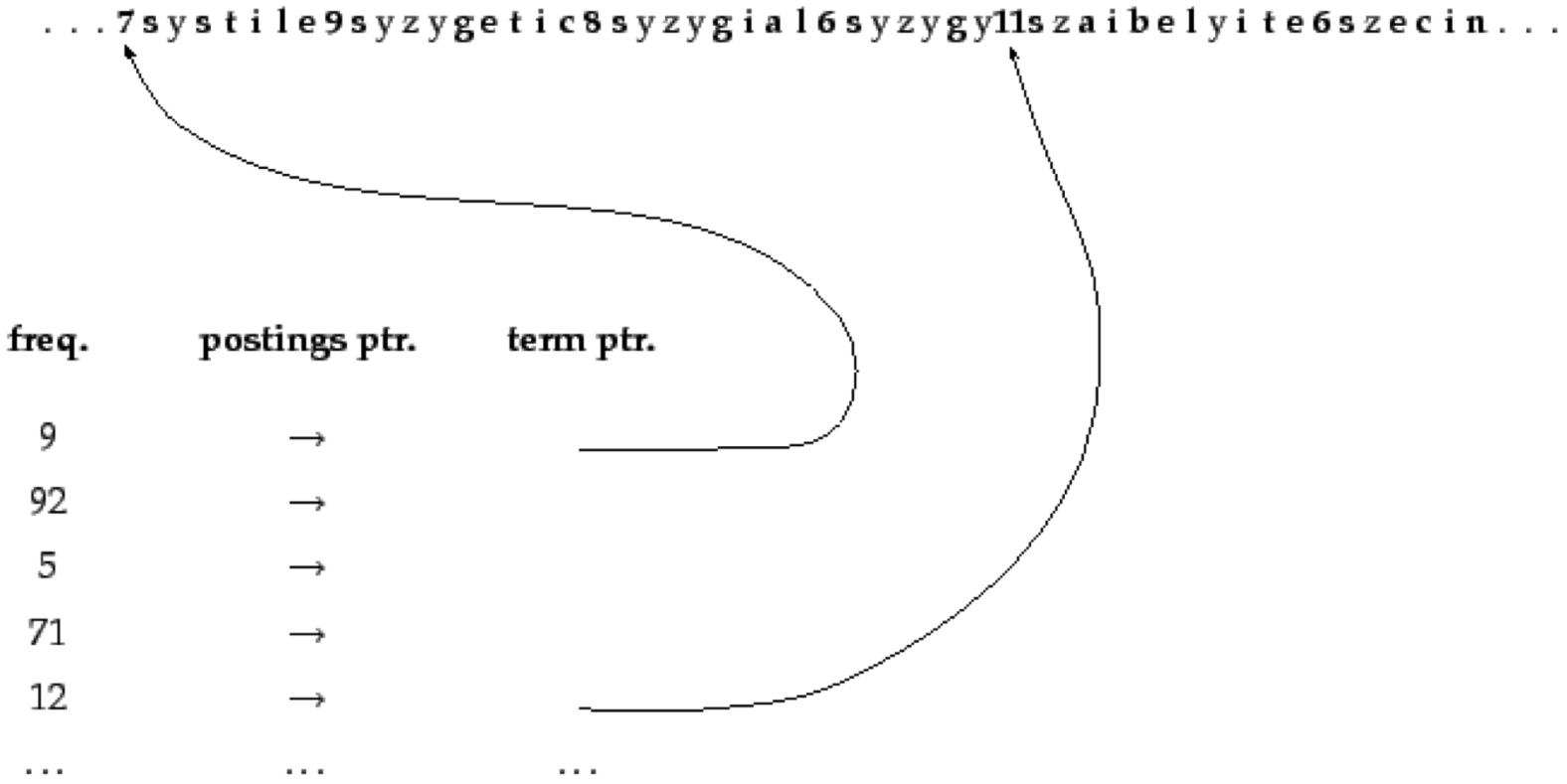
Compress Dictionary Terms

- Opportunities
  - Average English words only have 8 characters, so waste 12 character space in average for each word
- Solutions
  - View dictionary terms as a string
  - 400,000 words only need 3 bytes for term pointer
    - How this is determined?
  - Many other solutions
    - See IIR chapter 5



► **Figure 5.4** Dictionary-as-a-string storage. Pointers mark the end of the preceding term and the beginning of the next. For example, the first three terms in this example are systile (frequency 9), syzygetic (frequency 92) and syzygial (frequency 5).

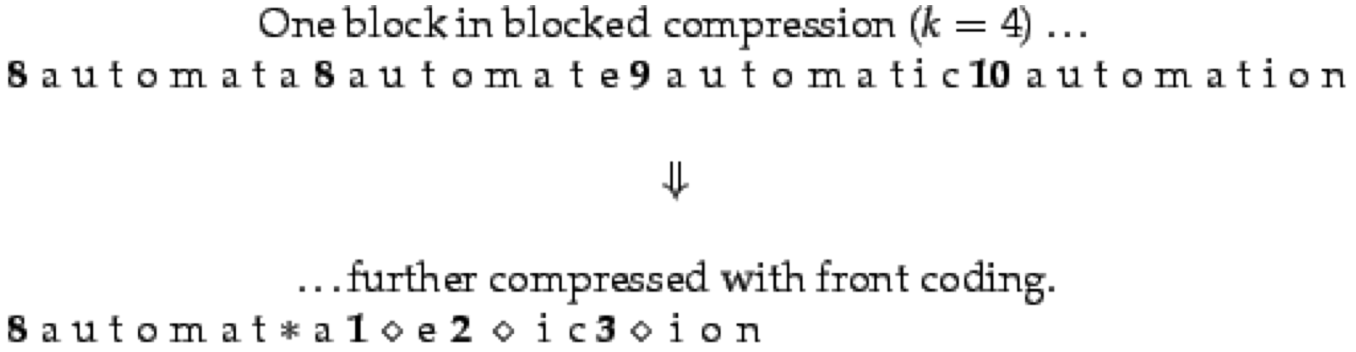
Blocked Storage



► **Figure 5.2** Blocked storage with four terms per block. The first block consists of systile, syzygetic, syzygial, and syzygy with lengths 7, 9, 8 and 6 characters, respectively. Each term is preceded by a byte encoding its length that indicates how many bytes to skip to reach subsequent terms.

Front Coding

- Consecutive entries in an alphabetically sorted list share common prefixes => front coding to further compress



► Figure 5.3 Front coding. A sequence of terms with identical prefix ("automat") is encoded by marking the end of the prefix with * and replacing it with ◇ in subsequent terms. As before, the first byte of each entry encodes the number of characters.

Effects of Dictionary Compression

- Reduce the dictionary size from 11.2MB to 5.9MB

representation	size in MB
dictionary, fixed-width	11.2
dictionary, term pointers into string	7.6
$\sim$, with blocking, $k = 4$	7.1
$\sim$, with blocking & front coding	5.9

Compress Posting Lists

- Opportunities
  - Distribution of numbers is skewed
    - Most numbers are small (word locations, term frequency)
  - The longest lists take the most space, compressing them save the most space
- Solutions
  - Delta encoding
  - Unary code
  - Gamma code

Delta Encoding

Delta Encoding ("Storing Gaps")

- **Store the differences between numbers**
- **Reduces range of numbers.**
- **Produces a more skewed distribution.**
- **Increases probability of smaller numbers.**
- **(Stemming also increases the probability of smaller numbers.)**

No Delta Encoding

Doc ID	121
TF	3
Loc	18
Loc	47
Loc	68
DocID	135
TF	2
Loc	22
Loc	35

Delta Encoding

Doc ID	121
TF	3
Loc	18
Loc	29
Loc	21
DocID	14
TF	2
Loc	22
Loc	13

Variable Length Codes

- **Store first 2^7 numbers in 7 bits: 1xxxxxx**
- **Store next 2^{14} numbers in 14 bits: 0xxxxxx1xxxxxx**
- **Store next 2^{21} numbers in 21 bits: 0xxxxxx0xxxxxx1xxxxxx**
- **And so on….**
- **Often used on inverted lists, after delta encoding integer data**
  - Many numbers fit in one byte
  - It is rare to exceed two bytes (16,511)
- **Advantages:**
  - Effective, non-parametric
  - Encoding and decoding can be done very efficiently
  - Easy to find number boundaries without decoding

Gamma Code (γ code) - I

- Usages
  - Used when the largest encoded value is not known ahead of time, or
  - to compress data in which small values are much more frequent than large values.

Gamma Code (γ code) - 2

- Unary code
  - A number N_{10} is coded with N one then is followed by a zero
    - E.g. 3 is coded as 1110, 7 is coded as 11111110.
  - 1 and 0 can be switched in unary code
    - So, 3 can be coded as 0001, 7 as 00000001
- Gamma code contains two parts *length* and *offset*
  - *Offset* is in binary, but with the leading 1 removed. For example, for 11 (binary 1011) *offset* is 011.
  - *Length* encodes the length of *offset* in unary code. For 11, the length of *offset* is 3 bits, which is 1110 in unary.
  - Now gamma code is the concatenation of length and offset. so for 11, its gamma code is 1110011

Compression of Index

representation	size in MB
dictionary, fixed-width	11.2
dictionary, term pointers into string	7.6
$\sim$, with blocking, $k = 4$	7.1
$\sim$, with blocking & front coding	5.9
collection (text, xml markup etc)	3600.0
collection (text)	960.0
term incidence matrix	40,000.0
postings, uncompressed (32-bit words)	400.0
postings, uncompressed (20 bits)	250.0
postings, variable byte encoded	116.0
postings, γ encoded	101.0

► **Table 5.6** Index and dictionary compression for Reuters-RCV1. The compression ratio depends on the proportion of actual text in the collection. RCV1 contains a large amount of XML markup. Using the two best compression schemes, γ encoding and blocking with front coding, the ratio compressed index to collection size is therefore especially small for RCV1: $(101 + 5.9)/3600 \approx 0.03$.

So

- We have discussed:
  - The data structure for indexing documents
  - The data structures for accessing the index
  - Methods for compressing the inverted index
- Most of our discussions focus
  - Efficiency: how quickly and accurately build and access index
  - We have not talk about the effectiveness of retrieval
    - We will talk about this in our retrieval models

Assignment 2: index construction

- Due Feb 19
- Task : construct a simple index
- Build on top of the outcomes of assignment 1
- More information is in courseweb