

605.744 Information Retrieval

Index Construction



JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

Outline

- Brief Review
- Tolerant Retrieval
 - Supporting wildcards
 - Handling misspellings and lexical variation
 - Alternate tokenization: character n-grams
- Index Construction
 - Basic Dictionary Representations
 - In memory
 - Using external storage

Week 1 Topics

- Boolean Representation
- Inverted Files (high level)
- Processing text to create indexing terms
- Program #1: Corpus Statistics
- Chapters 1, 2
 - NOTs, Skip lists, Positional index, stopping

Boolean Operations

- Queries composed of these operators
 - AND (conjunction)
 - OR (disjunction)
 - NOT (negation)
- We covered:
 - How to compute result sets
 - Intersection / Merging algorithm
 - How to optimize query execution time for multiple term queries
 - Pros & Cons of Boolean Search

Terminology (1)

- Indexing Term

- The sub-atomic particles of documents. In practice, terms are essentially words.
- However, terms can be stemmed words, multi-word phrases, or other strings.

- Vocabulary Size

- The number of unique indexing terms
- Sometimes call “dictionary size”
- 100,000 to 1 million is a normal figure; large Internet engines might use 20 million terms

Terminology (2)

- Corpus
 - Alternative term for document collection
- Docid
 - Unique integer representing each document

Terminology (3)

- Document Frequency
df(term) – the number of documents that a term appears in
- Collection Frequency / Corpus frequency
cf(term) – the number of times a term appears in a collection
 - For example, “the” appears in some documents more than once
- Term Count or Term Frequency
tf(term, docid) – for a particular document, the number of times that a given term appears in it

Terminology (4)

- Dictionary
 - The part of an index that stores information about a term, except for which specific documents the term is present in
 - Generally the dictionary is a list of words, their document frequency, and a pointer to a postings list
- Posting or Postings Entry
 - A posting is a tuple that indicates how many times a term appears in a document
 - (doc5, 3) indicates that this term appears 3 times in docid # 5
- Inverted File / Inverted Index
 - A set of postings lists for terms
 - Lets one quickly determine which documents contain which terms

Tokenization

■ Stopwords

- Common words like articles, prepositions, pronouns, etc...
- Usually lists of 100 – 300 words
- Considered unimportant for finding documents, so often removed from an index

the, and, of, with, ...

■ Stemming

- Transforming related word forms into a common representation
- Usually by identifying a root form or stem
- Enables finding documents which use a variant form

swimming, swimmer, swims -> swim

tables, table, labelled -> tabl

From Chapter 2 in IIR

- Skip lists
- Phrase index
- Biword index
- Positional index

Tolerant Retrieval

- Wildcard queries
 - Permuterm dictionary
 - k-grams (or n-grams or q-grams)
- Spelling Correction
 - General purpose
 - Proper names: Soundex algorithm
- Character N-gram Tokenization (not in IIR)

Matching User Input

- Indexing terms from documents may not match user query terms
- Two common reasons
 - Lexical variation (e.g., building, builder, buildings; juggler, juggle, juggling, juggled)
 - Misspellings
- Sometimes users prefer a shorthand where they can indicate all words that match a pattern
 - Wildcard matching is used where '*' indicates any string
 - **compa*** would match **company**, **companies**, and **companion**, among other terms

Wild-card queries: *

- mon^* : find all docs containing any word beginning “mon”.
- Easy with binary tree (or B-tree) lexicon: retrieve all words in range: $\text{mon} \leq w < \text{moo}$
- $^*\text{mon}$: find words ending in “mon”: harder
 - Maintain an additional B-tree for terms backwards.
 - Can retrieve all words in range: $\text{nom} \leq w < \text{non}$.

Exercise: from this, how can we enumerate all terms meeting the wild-card query *pro*ent*?

Query processing

- At this point, we have an enumeration of all **terms** in the dictionary that match the wild-card query.
- To identify matching documents we still have to look up the postings for each enumerated term.
- E.g., consider the query:

*se*ate AND fil*er*

This may result in the execution of many Boolean *AND* queries. (Or many OR queries)

(senate OR seagate OR sedate ...) AND (filter OR filibuster OR filer ...)

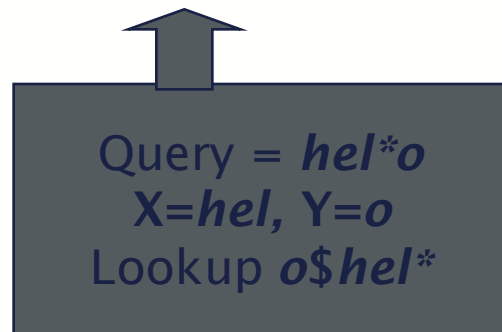
*'s inside a query term

- How can we handle '*'s in the middle of query term?
 - With one lookup
 - (Especially multiple '*'s)
- The solution: transform every wild-card query so that the '*'s occur at the end
- This gives rise to the Permuterm Index.

“index” is an unfortunate name here. This is really a mapping from a partial term to a set of matching terms. No documents are indexed in a permuterm index.

Permuterm index

- For term *hello* index under:
 - *hello\$, ello\$h, llo\$he, lo\$hel, o\$hell*
where \$ is a special symbol.
- Queries:
 - X lookup on X\$ X* lookup on X*\$
 - *X lookup on X\$* *X* lookup on X*
 - X*Y lookup on Y\$X* X*Y*Z ???



Permuterm query processing

- Rotate query wild-card to the right
- Now use B-tree lookup as before.
- *Permuterm problem: \approx quadruples lexicon size*

1. Wildcard usage is rare
2. Permuterm Indexes take up more RAM
3. Two lookups (with forward and reverse binary trees) isn't that bad

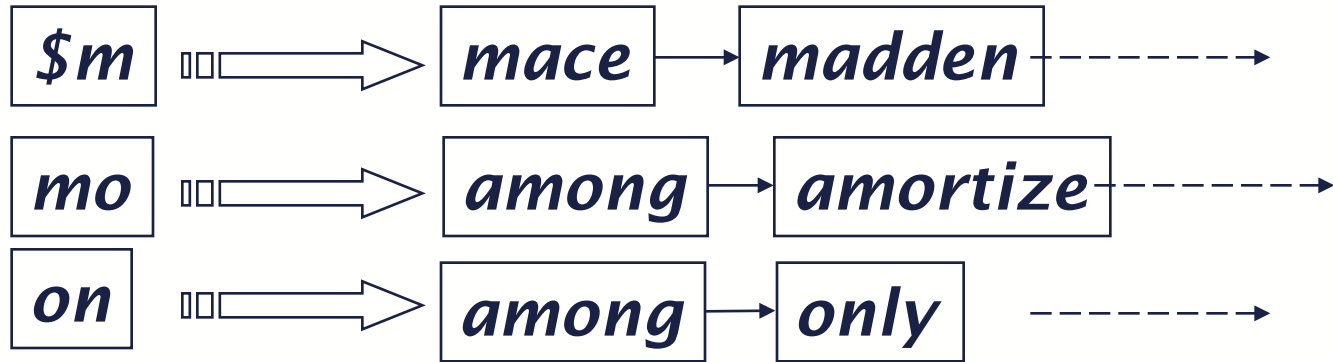
k-gram conversion

- Enumerate all k -grams (sequence of k chars) occurring in any term
- e.g., from text “*April is the cruelest month*” we get the 2-grams (*bigrams*)

\$a,ap,pr,ri,il,l\$,\$i,is,s\$,\$t,th,he,e\$,\$c,cr,ru,
ue,el,le,es,st,t\$, \$m,mo,on,nt,h\$

- \$ is a special word boundary symbol
- Maintain a mapping from bigrams to dictionary terms that match each bigram.

k-gram example



Wild-cards with character k-grams

- Query *mon** can now be expanded by combining bigrams
 - *\$m WITH mo WITH on*
- Fast, space efficient
- Gets terms that match the n-gram version of our wildcard query.
- But we'd generate *moon*.
 - Must post-filter these terms against query.
- Surviving enumerated terms are then looked up in the term-document inverted index.

Spell correction

- Two principal uses
 - Correcting document(s) being indexed
 - Retrieve matching documents when query contains a spelling error

- Two main flavors
 - Isolated word

Check each word on its own for misspelling

Will not catch typos resulting in correctly spelled words e.g., from → form

- Context-sensitive

Look at surrounding words, e.g., I flew form Heathrow to Narita.

1990s Telecoms

- In the early 1990s, MCI took out a 1-800 number to attract collect calling business: 1-800-COLLECT



- AT&T was losing business, so they took out a rival number: 1-800-OPERATOR



- MCI (brilliantly), obtained the number: 1-800-OPERATER, a common misspelling
 - Many customers called the MCI "-ER" number, and AT&T lost some business

Document correction

- Primarily for OCR'ed documents
 - Correction algorithms tuned for this
- Goal: the index (dictionary) contains fewer OCR-induced misspellings
- Can use domain-specific knowledge
 - E.g., OCR can confuse O and D more often than it would confuse O and I (adjacent on the QWERTY keyboard, so more likely interchanged in typing).

Example OCR failure

Central Intelligence Agency
Office of Current Intelligence

14 July 1953

The attached article, "The Doctors' Plot," is the first in a series of working papers prepared by the staff of Project CAESAR. Project CAESAR was established by the Director of Central Intelligence to study all available information on the members of the Soviet hierarchy, the middle ranks as well as the higher

On the whole, the biographic information under scrutiny, like most categories of information on the Soviet Union, is inconclusive and frequently contradictory. The work of Project CAESAR has served, however, to stimulate reconstruction of developments and events affecting the Kremlin hierarchy, of which the first to be issued is "The Doctors' Plot". It will be followed by articles dealing with the subsequent and preceding periods.

is inconclusive and frequently contradictory.

Declassified CAESAR document obtained from:

<http://www.foia.cia.gov/collection/caesar-polo-and-esau-papers>

~~TOP SECRET~~

CAESAR

Central Intelligence Agency
Office of Current Intelligence
14 July 1953

MEMORANDUM

The attached article, "The Doctors' Plot," is the first in a series of working papers prepared by the staff of Project CAESAR. Project CAESAR was established by the Director of Central Intelligence to study all available information on the members of the Soviet hierarchy, the middle ranks as well as the higher

On the whole, the biographic information under scrutiny, like most categories of information on the Soviet Union, is inconclusive and frequently contradictory. The work of Project CAESAR has served, however, to stimulate reconstruction of developments and events affecting the Kremlin hierarchy, of which the first to be issued is "The Doctors' Plot". It will be followed by articles dealing with the subsequent and preceding periods.

Query mis-spellings

- Our principal focus here
 - E.g., the query *Alanis Morisett*
- We can either
 - Retrieve documents indexed by the correct spelling, OR
 - Return several suggested alternative queries with the correct spelling

Did you mean ... ?

In case you're curious, it is: Alanis Morissette

Isolated word correction

- Fundamental premise – there is a lexicon from which the correct spellings come

- Two basic choices for this

- A standard lexicon such as

Webster's English Dictionary

Or, a hand-maintained “industry-specific” lexicon

- The lexicon of the indexed corpus

E.g., all words on the web, including the mis-spellings

Isolated word correction

- Given a lexicon and a character sequence Q , return the words in the lexicon closest to Q
- What's "closest"?
- There are several alternatives
 - Edit distance
 - Weighted edit distance
 - n -gram overlap

Edit distance

- Given two strings S_1 and S_2 , the minimum number of basic operations to convert one to the other
- Basic operations are typically character-level
 - Insert
 - Delete
 - Replace
- E.g., the edit distance from *swims* to *swam* is 2
- Generally found by dynamic programming
- Also called “Levenshtein distance”

Weighted edit distance

- As above, but the weight of an operation depends on the character(s) involved
 - Meant to capture keyboard errors, e.g. m more likely to be mistyped as n than as q
 - Therefore, replacing m by n is a smaller edit distance than by q
 - (Same ideas usable for OCR, but with different weights)
- Require weight matrix as input
- Modify dynamic programming to handle weights

Edit distance to all dictionary terms?

- Given a (mis-spelled) query – do we compute its edit distance to every dictionary term?
 - Expensive and slow
- How do we cut the set of candidate dictionary terms?
- Here we use n -gram overlap for this

n-gram overlap

- Enumerate all the *n*-grams in the query string
- Use the *n*-gram index (recall wild-card search) to retrieve all lexicon terms matching any of the query *n*-grams
- Threshold by number of matching *n*-grams
 - Variants – weight by keyboard layout, etc.

Remember: n-grams == k-grams

Example with trigrams

- Suppose the text is *november*
 - Trigrams are *nov*, *ove*, *vem*, *emb*, *mbe*, *ber*.
- The query is *december*
 - Trigrams are *dec*, *ece*, *cem*, *emb*, *mbe*, *ber*.
- So 3 trigrams overlap (of 6 in each term)
- How can we turn this into a normalized measure of overlap?

One option – Jaccard coefficient

- A commonly-used measure of overlap
- Let X and Y be two sets; then the J.C. is

$$|X \cap Y| / |X \cup Y|$$

- Equals 1 when X and Y have the same elements and zero when they are disjoint
- X and Y don't have to be of the same size
- Always assigns a number between 0 and 1
 - Use a threshold to decide if you have a match
 - E.g., if J.C. > 0.8, declare a match

Dice Coefficient

- A similar metric is the Dice Coefficient
- It also varies from 0 (bad match) to 1 (perfect match)
 - See: https://en.wikipedia.org/wiki/Sørensen–Dice_coefficient

$$2|X \cap Y| / |X| + |Y|$$

Example

X=November: nov, ove, vem, emb, mbe, ber

Y=December: dec, ece, cem, emb, mbe, ber

$$Dice = 2 * 3 / (6 + 6) = 6 / 12 = 0.5$$

Context-sensitive spell correction

- Text: *I flew from Heathrow to Narita.*
- Consider the phrase query “*flew form Heathrow*”
- We’d like to respond

Did you mean “*flew from Heathrow*”?

because no docs matched the query phrase.

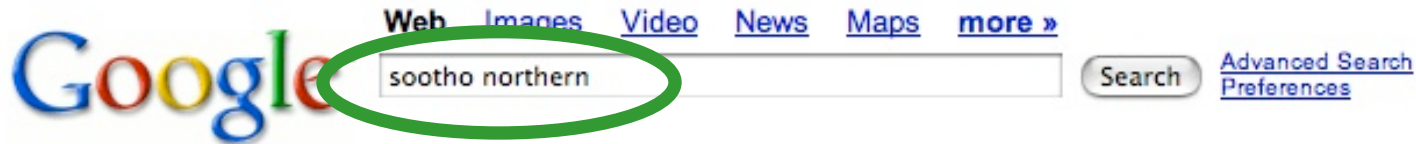
Issues in spell correction

- Will enumerate multiple alternatives for “Did you mean”
- Need to figure out which one (or small number) to present to the user
- Use heuristics
 - The alternative hitting most docs
 - Query log analysis + tweaking

For especially popular, topical queries

- Spell-correction is computationally expensive
 - Run only on queries that match few docs

northern sotho



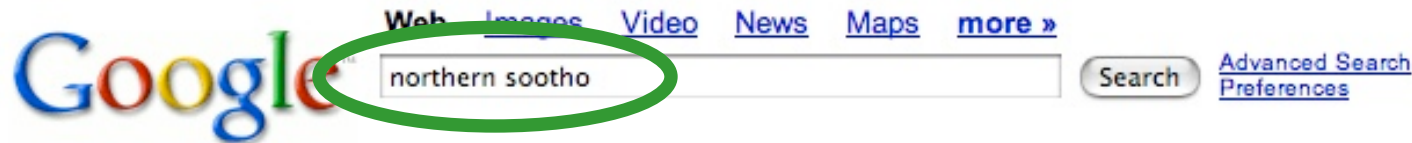
Web

[FDA Enforcement Report](#)

Selig Chemical Industries brand **Sootho** Food Processing Sanitizing Lotion Soap ... FILED
February 15, 1994; U.S. District Court for the **Northern** District of ...

www.fda.gov/bbs/topics/ENFORCE/ENF00307.html - 38k - [Cached](#) - [Similar pages](#)

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

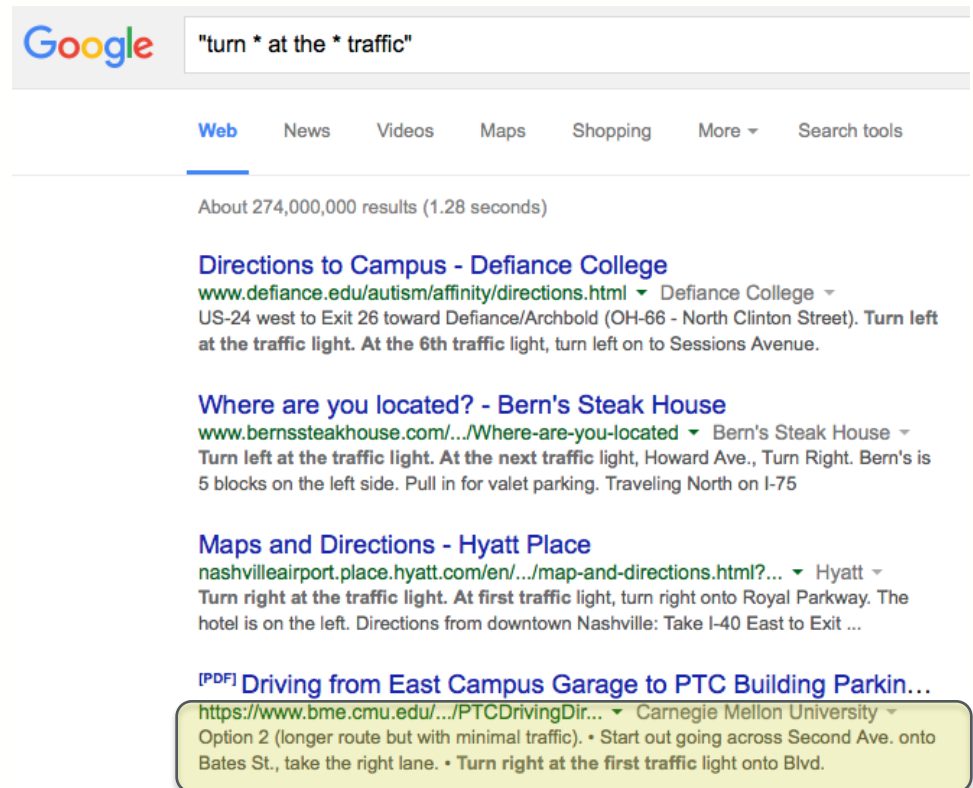


Web

Did you mean: [northern sotho](#)

Language Modeling Demo (KWIC)

- Examination of co-occurrence patterns can help inform spelling correction
 - <http://www.someya-net.com/concordancer/bigram.html>
- Google wildcard search
 - “turn * at the * traffic”
 - “graduated with * degree in
 - "died from a * wound"



The screenshot shows a Google search interface with the query "turn * at the * traffic" entered in the search bar. The results page displays the Google logo, navigation tabs (Web, News, Videos, Maps, Shopping, More, Search tools), and search statistics (About 274,000,000 results (1.28 seconds)). The top results are:

- Directions to Campus - Defiance College**
www.defiance.edu/autism/affinity/directions.html ▾ Defiance College ▾
US-24 west to Exit 26 toward Defiance/Archbold (OH-66 - North Clinton Street). Turn left at the traffic light. At the 6th traffic light, turn left on to Sessions Avenue.
- Where are you located? - Bern's Steak House**
www.bernssteakhouse.com/.../Where-are-you-located ▾ Bern's Steak House ▾
Turn left at the traffic light. At the next traffic light, Howard Ave., Turn Right. Bern's is 5 blocks on the left side. Pull in for valet parking. Traveling North on I-75
- Maps and Directions - Hyatt Place**
nashvilleairport.place.hyatt.com/en/.../map-and-directions.html? ▾ Hyatt ▾
Turn right at the traffic light. At first traffic light, turn right onto Royal Parkway. The hotel is on the left. Directions from downtown Nashville: Take I-40 East to Exit ...
- [PDF] Driving from East Campus Garage to PTC Building Parkin...**
<https://www.bme.cmu.edu/.../PTCDrivingDir...> ▾ Carnegie Mellon University ▾
Option 2 (longer route but with minimal traffic). • Start out going across Second Ave. onto Bates St., take the right lane. • Turn right at the first traffic light onto Blvd.

Soundex

- Class of heuristics to expand a query into phonetic equivalents
 - Language specific – mainly for names
 - E.g., *chebyshev* → *tchebycheff*
- Turn every token to be indexed into a 4-character reduced form
 - Do the same with query terms
- Build and search an index on the reduced forms
 - (when the query calls for a soundex match)

Soundex – typical algorithm

- Retain the first letter of the word.
- Change all occurrences of the following letters to '0' (zero):
'A', 'E', 'I', 'O', 'U', 'H', 'W', 'Y'.
- Change letters to digits as follows:
 - B, F, P, V \rightarrow 1
 - C, G, J, K, Q, S, X, Z \rightarrow 2
 - D, T \rightarrow 3
 - L \rightarrow 4
 - M, N \rightarrow 5
 - R \rightarrow 6

Soundex continued

- Replace consecutive identical digits with only the first, if they came from letters in the same category in the original string.
- Remove all zeros from the resulting string.
- Pad the resulting string with trailing zeros and return the first four positions, which will be of the form <uppercase letter> <digit> <digit> <digit>.
- E.g., Herman becomes H655.

Maryland resident? Try your last name and look at your driver's license number. The DMV uses Soundex. Fails if you had a license and then changed your name.

Word-based Information Retrieval

- Most traditional information retrieval systems index documents according to the words in those documents.
- Word-based retrieval is language-specific (e.g., a retrieval system for English will not work as well for Arabic, Japanese, Korean, Turkish, and other languages).
- Word-based retrieval performs poorly when the documents to be retrieved are garbled or contain spelling mistakes (e.g., from OCR or speech transcription).

Character N-gram Tokenization

- Represent text as overlapping substrings
- Fixed length of n of 4 or 5 is effective in alphabetic languages
- For text of length m , there are $m-n+1$ n-grams
- N-grams may span word boundaries

	s	w	i	m	m	e	r	s	
_	s	w	i	m					
	s	w	i	m	m				
		w	i	m	m	e			
			i	m	m	e	r		
				m	m	e	r	s	
					m	e	r	s	_

N-grams as Indexing Terms

- This is not a way to find matching terms for wildcards or a means for spelling correction, but rather a different way of indexing text
- Advantages: simple, address morphology, surrogate for short phrases, robust against spelling & diacritical errors, language-independent
- Disadvantages: conflation (e.g., simmer, polymers), more n-grams -- they can incur both speed and disk usage penalties

Four score and seven...

Words

four
score
and
seven
...

N-Grams

four_
our_s
ur_sc
r_sco
...

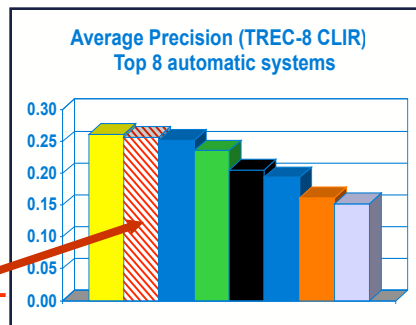
N-grams in non-Asian Languages

- N-grams are overlapping sequences of n characters

- 6-grams of “Johns Hopkins” include: Johns_, ohns_H, hns_Ho, ns_Hop, and so forth
- N-grams have been used chiefly in Asian languages where word identification is difficult

- TREC-8 Cross-Language Task

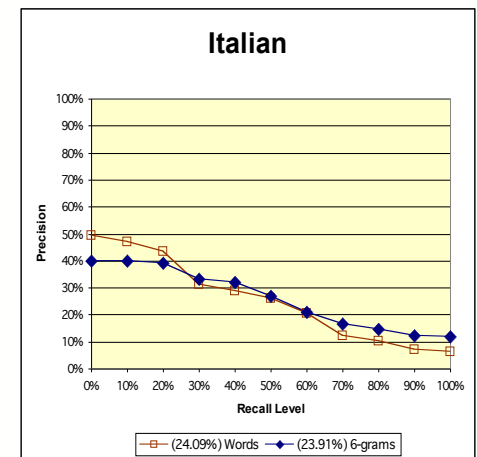
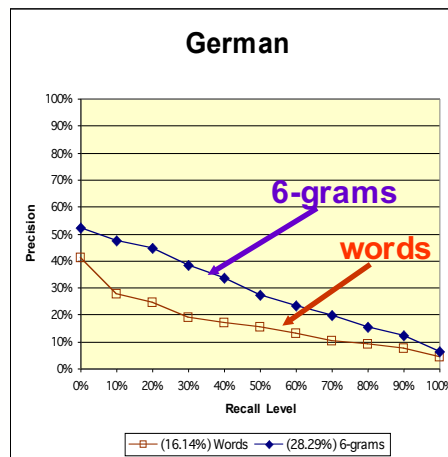
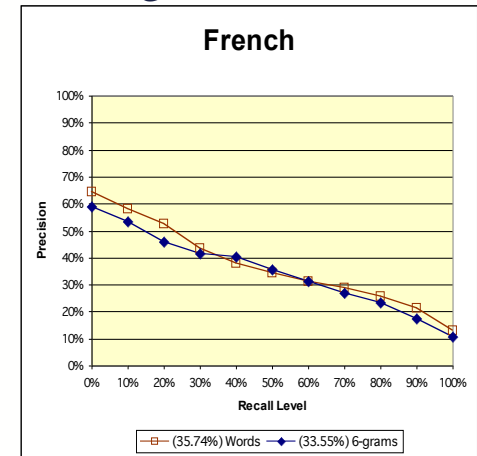
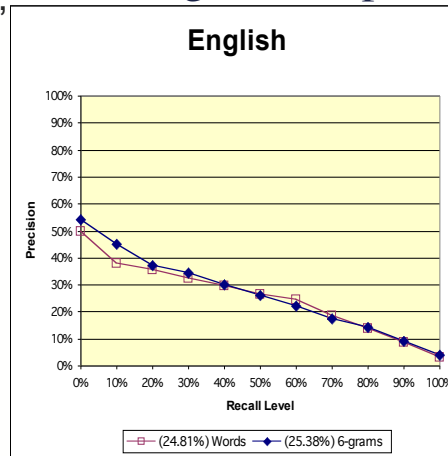
- English, French, German, Italian



- N-grams and words comparable

- But German words worse than 6-grams due to compounding
- “einzelnefamilienstadtwohnung” vs. “single family townhouse”

Monolingual comparison of n-grams and words



Chapter 4: Index Construction

- Basic Dictionary Representations
- Indexing methods
 - Inverted File Algorithms

Study IIR 4.1, 4.2, 4.3, Skip 4.4, Read 4.5-4.7

- Dynamic Indexes (4.5)

Sample Text

- Doc 1: Socrates is a man
- Doc 2: All men are mortal
- Doc 3: Socrates is mortal, mortal

Dictionary

Term	TermID	DF	#Occur	Pointer
socrates	0	2	2	
is	1	2	2	
a	2	1	1	
man	3	1	1	
all	4	1	1	
men	5	1	1	
are	6	1	1	
mortal	7	2	3	

How should the dictionary be represented?

- Sorted Array
 - slow to update
- Tree
- Tries
- Hashtables

Arrays

- Arrays provide access to a set of items
 - Items in the array are stored contiguously in memory
 - Changing the length of an array is generally expensive (portions must be copied)
 - Length of arrays is fixed (known)
- Fast access to data
 - Reading or setting the i^{th} item is $O(1)$
 - No pointers are required
 - Contiguous data may reduce paging if virtual memory is being used (compared to pointers)

fruit

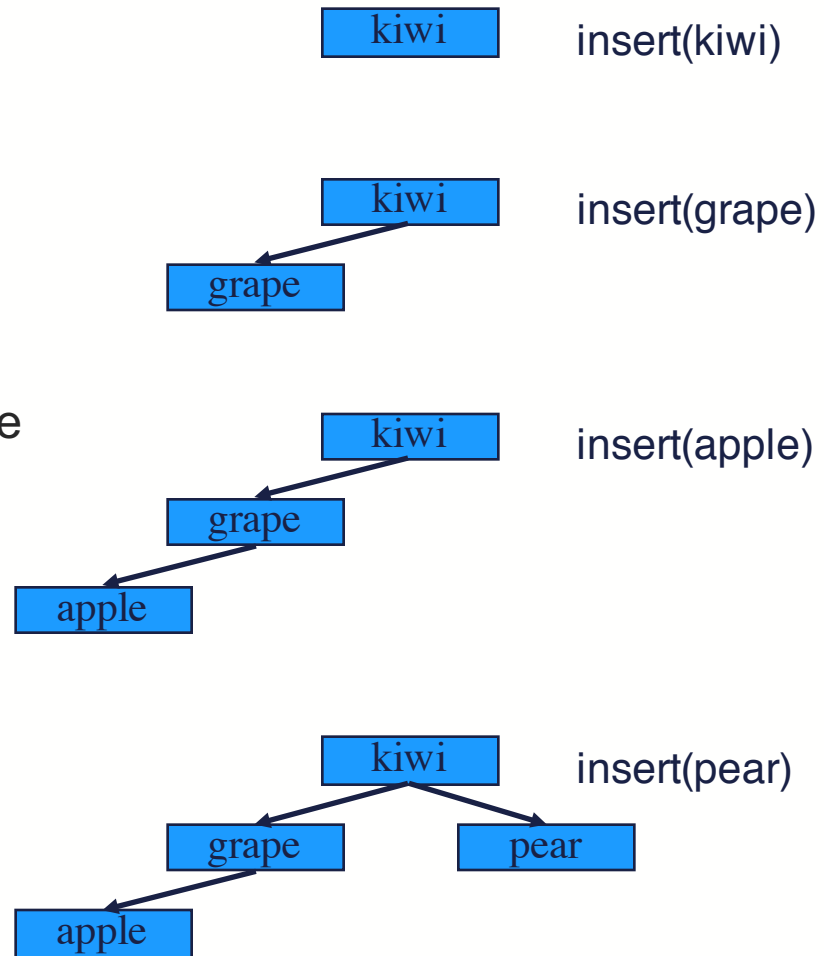
0	apple
1	kiwi
2	lemon
3	orange
4	peach
5	pear

Binary Trees

- Generally Lists and Arrays require linear time to search for a given item
 - If items can be ordered much better is possible (just like a phone book)
 - Insertion and Deletion become $O(\log n)$ operations
 - Search becomes $O(\log n)$
 - Data is preserved in sorted order, thus there is no need to sort the data
- Tree implementation
 - 2 pointers & 1 item per node
 - Trees can become unbalanced

Special approaches maintain balance

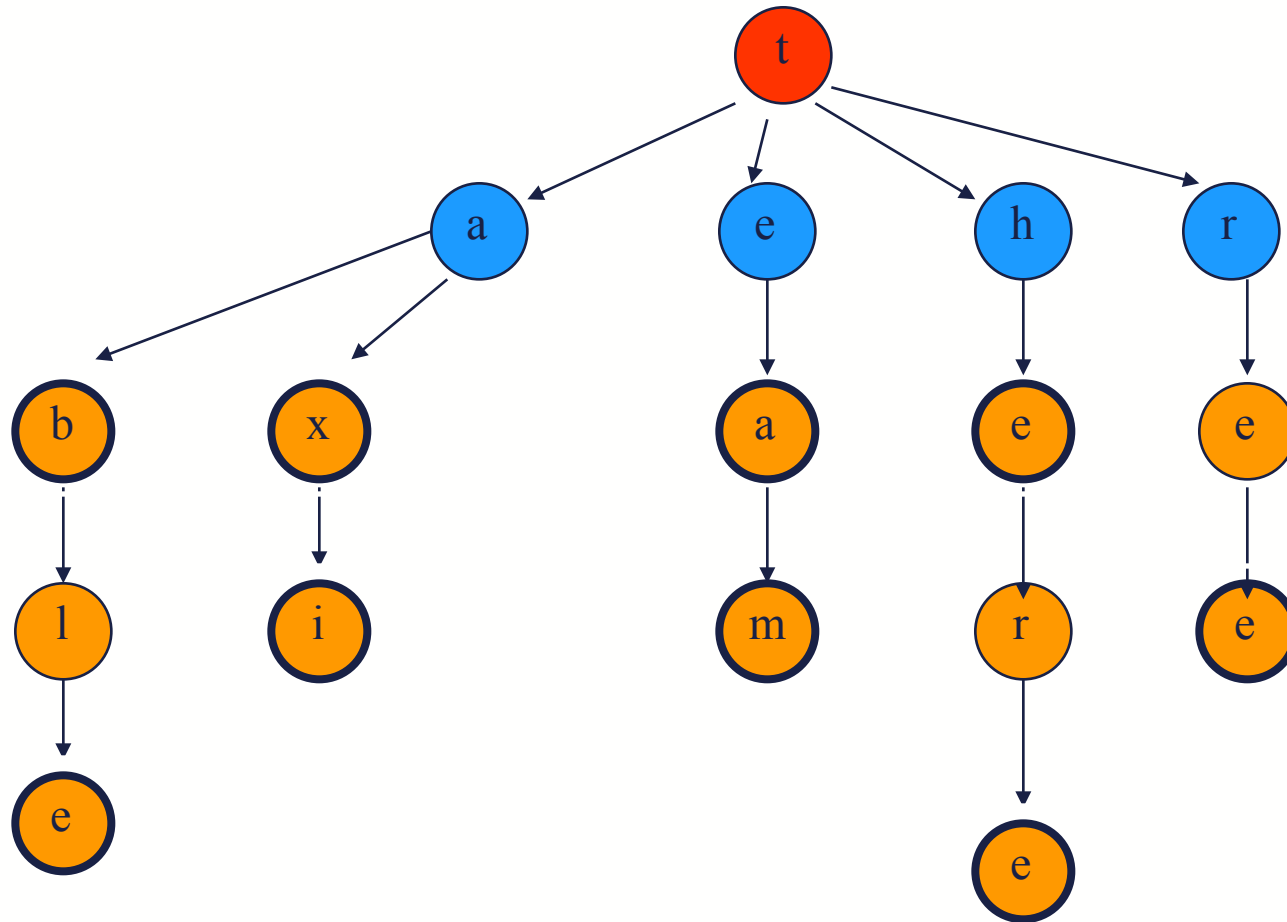
Red-Black trees, AVL trees



Tries (de la Briandais - 1959)

- Given an ordered alphabet, represent words on a lexicographic tree
 - Root of tree is first character in word
 - Children of root represent second character, and so on
 - Height of tree is the number of nodes on the longest path from any leaf to the root (i.e., length of longest word)
 - During search $O(\log_{\text{alphabet}} n)$ internal nodes examined, worst case
- Variants
 - Patricia Trees (collapse single child nodes)
 - Suffix tree

A Trie



How should each level (i.e., the blue nodes) be stored?

What changes if we add the word “teams” to the dictionary?

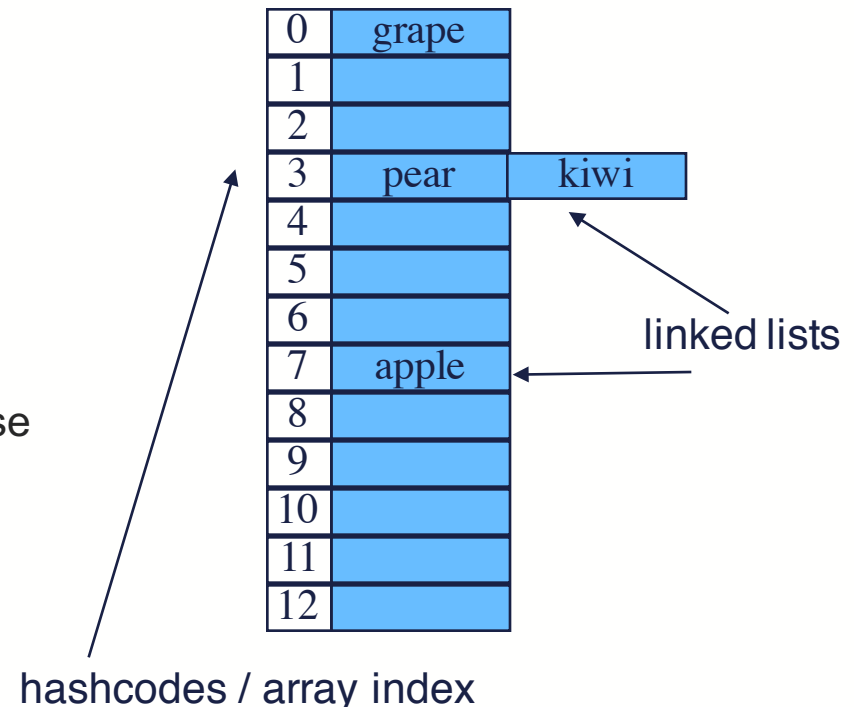
Hashtables

- Logarithmic access is fairly efficient
 - Search in a tree with 1,000,000 entries requires only 20 comparisons
- $O(1)$ time is possible, if
 - Order information is unimportant
 - Data values are unique

that is, duplicates are not maintained

 - Data items can be tested for equality
 - Items can be mapped to the natural numbers using a 'hash' function
- Table implementation
 - An array of size much smaller than the universe of possible items (keys)
 - A value may be stored with each key
 - Many variants

Item	Code
apple	7
kiwi	3
pear	3
grape	0



Guidance

- Easy to implement...
 - Hashtables, Trees, & Tries
 - Each of these is reasonable for the smallish collections we will use in class
- Minimal memory (next module)
 - Front-coding (and arrays, or B-trees)
 - Perfect hash functions

Lexicon payload

- Must include
 - term itself
 - pointer to inverted file location
- Could keep only on disk, but useful in memory
 - document frequency
 - We don't normally store postings in memory

They go into the inverted file

- Don't strictly need
 - termid, number of occurrences
- How to store the dictionary on disk
 - Binary data, length-encoded strings

Building Inverted Files

- Doc 1: Socrates is a man
- Doc 2: All men are mortal
- Doc 3: Socrates is mortal, mortal

Dictionary

Term	ID	DF	#Occur	Pointer
socrates	0	2	2	
is	1	2	2	
a	2	1	1	
man	3	1	1	
all	4	1	1	
men	5	1	1	
are	6	1	1	
mortal	7	2	3	

Inverted File

Doc	times	Doc	times
1	1	3	1
1	1	3	1
1	1		
1	1		
2	1		
2	1		
2	1		
2	1	3	2

Records in inverted file are pairs (docid and count)

Data structures usually rely on termids vs. strings

Notation (differs from text, see IIR 4.1)

- t : the size of the text, in words
- N : the number of documents
- v : vocabulary size
- k : the average number of *unique* terms per document
 - $t > k * N$ *but $O(k*N) = O(t)$, usually*
 - $t \ll v * N$
- M : the amount of main memory available

Space Requirements

- The space required for the **vocabulary** is rather small. According to *Heaps'* law the vocabulary, **v**, grows as $O(t^\beta)$, where β is a constant between 0.4 and 0.6 in practice
- On the other hand, the occurrences (**inverted file**) demand much more space. Since each word appearing in the text is referenced once in that structure, the space is $O(t)$

Plan A: Memory-based inversion

- All the vocabulary is kept in a suitable data structure
 - For each word also store a list of its occurrences in documents
 - List of (docid, term count) pairs
- Each word of the text is read and searched in the vocabulary
- If it is not found, it is added to the vocabulary with a empty list of occurrences and the new tuple is added to the end of its list of occurrences
- Count duplicates in a document

Method A cont'd

- Once the text is exhausted the vocabulary is written to disk with the list of occurrences. Two binary files are created:
 - In the first file, the list of occurrences are stored contiguously
 - In the second file, the vocabulary is stored in lexicographical order and, for each word, a pointer to its list in the first file is also included. This allows the vocabulary to be kept in memory at search time
- The overall process is $O(t)$ worst-case time
 - $O(t)$ scanning text, $O(t)$ lookups*, $O(t)$ writing to disk
- **Fails** if we cannot fit the entire inverted file in memory at once

Algorithm B (sort-based inversion)

- As soon as each term is seen, a tuple (or record) is written to a *temporary* file
 - <socrates, doc1, 1>
 - <men, doc2, 1>
 - <socrates, doc3, 2>
- These intermediate files are externally sorted *in-place*, on disk
 - term is primary key, docid is secondary key
 - Assume each temp. file is of size M
- Then the sorted files are merged together

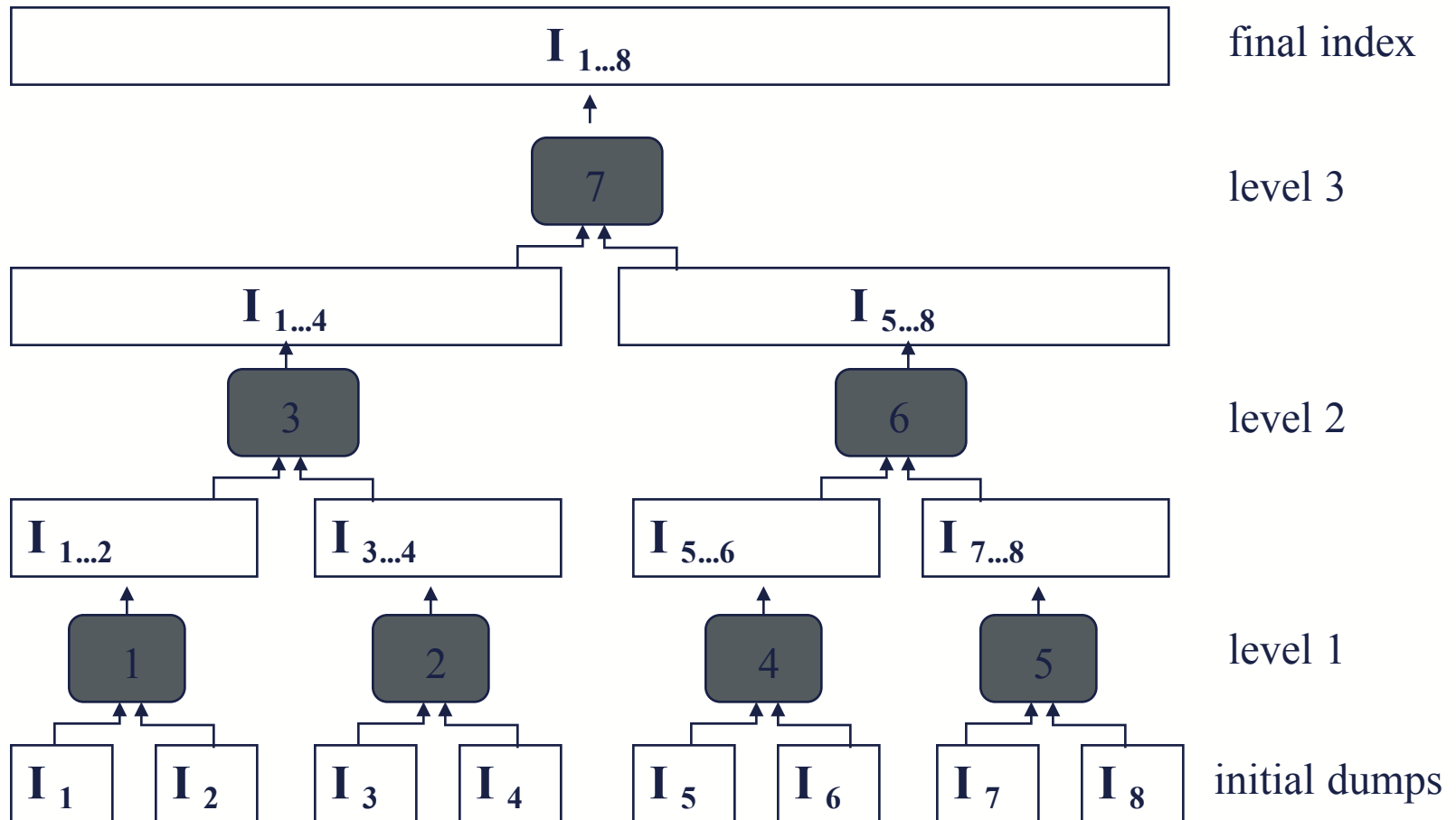
Alg. B - Analysis

- Performance is not great
 - lots of I/O (including random access)
 - sorting $C * O(t * \log(t/M))$ C is big
 - merging $O(t)$
- What is in memory?
 - Only the dictionary
- Works if memory is very limited

Alg C (memory-sorted inversion)

- An option is to fill main memory (M) with tuples, and then, sort the tuples and write the partial index I_i obtained up to now to disk. Then the current, partial index is erased from main memory and we continue with the rest of the text.
 - This nicely avoids sorting on disk
 - Described in [IIR 4.2](#) (and in MG 5.2)
- Once the text is exhausted, a number of partial indices I_i exist on disk
- The partial indices are merged to obtain the final index

Alg C: Hierarchical Merge Step



Merging Partial Inverted Files

- Doc 1: Socrates is a man
- Doc 2: All men are mortal
- Doc 3: Socrates is mortal, mortal

Partial inverted file (1+2)

Term	Doc	times
socrates	1	1
is	1	1
a	1	1
man	1	1
all	2	1
men	2	1
are	2	1
mortal	2	1

Inverted File (3)

Term	Doc	times
socrates	3	1
is	3	1
mortal	3	2

Here records sorted by termid
(socrates is termid 0, mortal is 7)

Analysis of Algorithm C

- The total time to scan the text is $O(t)$
- The number of partial indices is $O(t/M)$
 - Each is sorted at a cost of $O(M \log(M))$
- $\log_2(t/M)$ merging levels are required to merge the $O(t/M)$ partial indices
 - *Cost: $O(t * \log(t/M))$*
- The total cost of this algorithm is:
 - $O(t) + O(t/M * (M \log M)) + O(t * \log(t/M)) =$
 - *The larger of: $O(t * \log(M))$ or $O(t * \log(t/M))$*
- *Can we do better?*

Considered Pause...

- So far we've seen two kinds of algorithms
 - Alg A avoided sorting, but was not robust if memory was insufficient
 - Algs B,C **sort sub-indexes** and use **hierarchical merging** to avoid memory limits
- We can make two improvements
 - Better merging
 - Avoid sorting sub-indexes at all

(This approach is not clearly described in the text)

Algorithm D

- Alg C, but instead of performing a hierarchical merge, perform an *n-way merge*
- Described in *IIR, page 65 (a bit obliquely)*
 - “To do the merging, we open all block files simultaneously, and maintain small read buffers for the ten blocks we are reading and a write buffer for the **[one]** final merged index we are writing.
- Limitation is the number of open file handles
 - Some systems limit the number to 40-1024 or so open files (an operating system resource)
- Merging is more efficient
 - Overall performance is $O(t * \log(M))$

Algorithm E (IIR 4.3 is similar)

- Act like Alg A. When memory is exhausted, don't give up. Instead write a partial index out for the currently examined documents
- Keep the to-date lexicon in memory, but flush all of the postings out. Now start with the next set of documents
- Perform an n-way merge on the partial indexes
- Overall cost
 - Scan each word, add individual $\langle \text{docid}, \text{count} \rangle$ pair to variable length array associated with lexicon: $O(t)$
 - Write out each word to a partial index (once): $O(t)$
 - N-way merge: $O(t)$
 - Total: $3 * O(t) = O(t)$ *linear in length of text*

Example

- What will the dictionary / inverted file look like for these documents (use Algorithm A)

Assume all integers can be stored in 1 byte

Write out inverted file to disk in order A to Z

- all boy cows deserves eat every fudge good grass
- Doc 1: every good boy deserves fudge
- Doc 2: all cows eat all grass
- Doc 3: good boy deserves fudge
- Doc 4: good boy deserves all fudge

Why not hit “pause” and work this out?

Questions to Consider

- Does order of records in inverted file matter? Do they have to be written out in order from A to Z?
- What must happen to create a list of all documents containing the word fudge?
- How much space (in bytes) does the inverted file take up?

Inverted File Exercise: Solution

Doc 1: every good boy deserves fudge
 Doc 2: all cows eat all grass
 Doc 3: good boy deserves fudge
 Doc 4: good boy deserves all fudge

Dictionary

Term	DF	Offset
all	2	0
boy	3	4
cows	1	10
deserves	3	12
eat	1	18
every	1	20
fudge	3	22
good	3	28
grass	1	34

every	(1,1)		
good	(1,1)	(3,1)	(4,1)
boy	(1,1)	(3,1)	(4,1)
deserves	(1,1)	(3,1)	(4,1)
fudge	(1,1)	(3,1)	(4,1)
all	(2,2)	(4,1)	
cows	(2,1)		
eat	(2,1)		
grass	(2,1)		

Inverted File

2	2	4	1	1	1	3	1	4	1
2	1	1	1	3	1	4	1	2	1
1	1	1	1	3	1	4	1	1	1
3	1	4	1	2	1				

(docid, term count)

Solutions

- Does order of records in inverted file matter? Do they have to be written out in order from A to Z?
 - A: No, the order doesn't really matter. I just specified A to Z order so your answer would look like mine.
- What must happen to create a list of all documents containing the word fudge?
 - A: One first looks in the dictionary for the word fudge. If it is present, we look up its DF value which is 3. We then seek into the inverted file using the offset (22) and read 3 records. We find that fudge is in documents 1, 3, & 4.
- How much space (in bytes) does the inverted file take up?
 - A: 36 bytes