11-442 / 11-642:
Search Engines

Introduction to Search

Jamie Callan
Carnegie Mellon University
callan@cs.cmu.edu

Two Lecture Outline

A quick introduction to…

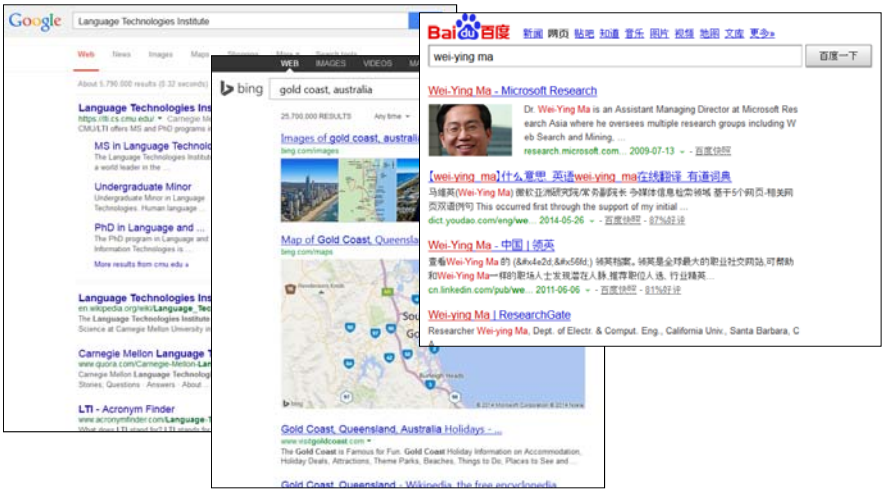
- Ad-hoc retrieval
- Information needs & queries
- Document representation
- Exact match retrieval
  - Unranked Boolean
  - Ranked Boolean
- Indexes
  - Inverted lists
- Document retrieval
  - TAAT
  - DAAT
- Query operators

Goal: Provide an overview of search ("the Big Picture")

- Later lectures explore these topics in greater detail

© 2019, Jamie Callan

Probably You are an Experienced Search Engine User



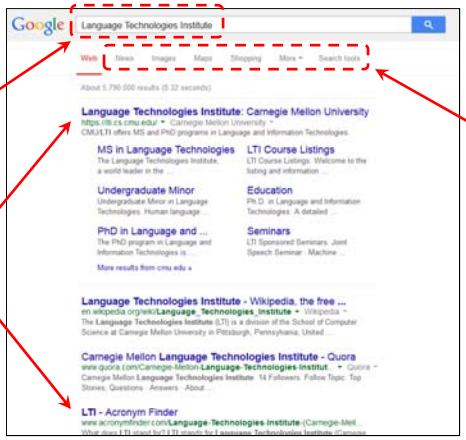
3

© 2019, Jamie Callan

How Does a Search Engine Developer or Researcher View the Search Process?

A query that expresses an information need

Documents (unstructured information)



Vertical search engines

4

© 2019, Jamie Callan

How Does a Search Engine Developer or Researcher View the Search Process?

A person starts with an information need

- The query is an approximate description of the information need

The person searches a corpus of unstructured information

- Documents

Goal: Find documents that satisfy the information need

- Search, retrieval

This lecture and the next present a simple end-to-end solution

- The "big picture"
- Later lectures go into more detail & more advanced material

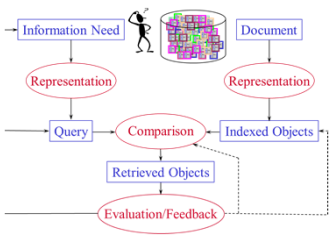
5

© 2019, Jamie Callan

Simple End-to-End Solutions

Requirements

- A way of representing information needs
- A way of representing document content
- A comparison or matching process



Initial solutions

- Boolean queries
- Exact-match retrieval models (unranked and ranked)

These solutions are primitive, but they are still used today

- Fast, easy to build, easy to understand

6

© 2019, Jamie Callan

Two Lecture Outline

A quick introduction to…

- Ad-hoc retrieval
- Information needs & queries
- Document representation
- Exact match retrieval
  - Unranked Boolean
  - Ranked Boolean
- Indexes
  - Inverted lists
- Document retrieval
  - TAAT
  - DAAT
- Query operators

Goal: Provide an overview of search ("the Big Picture")

- Later lectures explore these topics in greater detail

7

© 2019, Jamie Callan

Representing the Information Need

Exact-match retrieval models assume that a person can describe the information need as a Boolean query

- Relational database systems also make this assumption
- Most people are not good at creating Boolean queries
- Even well-trained people overestimate the quality of their queries

Examples:

- Angelina AND Jolie
- (Angelina AND Jolie) OR (Brad AND Pitt)
- (Angelina NEAR/2 Jolie) OR (Brad NEAR/2 Pitt)
  - NEAR/n is similar to a phrase operator
  - Match if terms are in this order, separated by a distance ≤ n

8

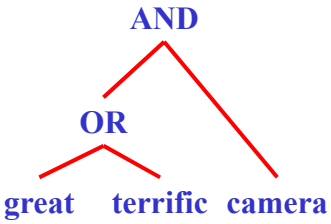
© 2019, Jamie Callan

Query Trees

Query: (great OR terrific) AND camera

Search engines represent the query as a tree

- Nodes are query operators
- Leaves are index terms



Does query q match document d?

- Use depth-first evaluation
- More about this later….

9

© 2019, Jamie Callan

Two Lecture Outline

A quick introduction to…

- Ad-hoc retrieval
- Information needs & queries
- Document representation
- Exact match retrieval
  - Unranked Boolean
  - Ranked Boolean
- Indexes
  - Inverted lists
  - Term dictionary
- Document retrieval
  - TAAT
  - DAAT
- Query operators

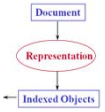
Goal: Provide an overview of search ("the Big Picture")

- Later lectures explore these topics in greater detail

10

© 2019, Jamie Callan

Representing the Document: An Example Document



A Great Choice.
Review by topjimmy5150
April, 21 2003

I have been looking and looking for a new camera to replace our bulky, but simple and reliable (but only fair picture taker) Sony Mavica FD73. My other choice (Besides the more expensive Nikon Coolpix 3100) was the (also more expensive) Sony Cybershot P72. I recommend any of these cameras, and I was set to buy the Sony, but at the last minute I cheaped out and bought the 2100. No regrets. I bought the camera (along with 128mb memory card (the stock 16mb card will be kept in the bag as a spare) and carrying case) at the new Best Buy in Harrisburg, PA. I also bought a set of 4 Nickle-Metal Hydride rechargable batteries and charger at Walmart for less than $20. I keep 2 in the camera and two in the charger/in the camera bag along with the original Lithium battery pack as spares.

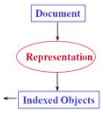
Hands down, the best feature of this camera is it's compact design. It is very small. My family likes to go camping during the summer, and last year we found the Mavica too

. . . .

(topjimmy5150, Epinions.com)

© 2019, Jamie Callan

Representing the Document



How should the contents of a document be represented?

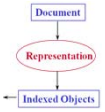
Today, assume that we will use words from the document

- Free-text indexing: Use just some of the words
  - Developed first, but … which words?
- Full-text indexing: Use most or all of the words
  - Most search engines do this

Later lectures consider other possibilities

© 2019, Jamie Callan

The Binary Full-Text Representation



Record which words occur in which documents

- Invented first
- A tabular representation is simple (but very inefficient)

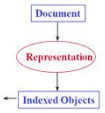
Vocabulary (Index Terms) |V|

Corpus |C|	a	abba	abend	ability	able	about	…	zooms
Doc_1	0	0	0	1	1	1	…	1
Doc_2	1	1	0	0	1	1	…	0
: : : :	:	: :	: :	: :	: :	: : :		: : :
Doc_n	0	0	1	1	0	1	…	0

Corpus: Document collection

© 2019, Jamie Callan

The Binary Full-Text Representation (The Bag of Words)

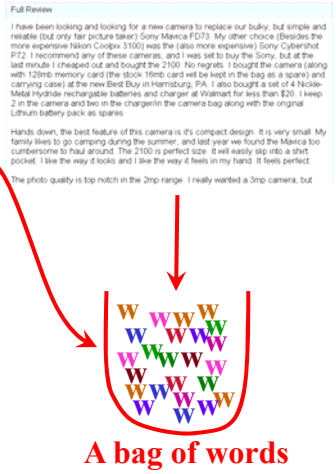


In the binary full-text representation, position and frequency are ignored

- The document is a "bag of words"
  - Or other features (covered later)

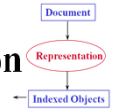
This is a simple representation of meaning

- But…surprisingly effective for search and other tasks (e.g., classification)



© 2019, Jamie Callan

Frequency-Based Full-Text Representation



Record the frequency of each word in each document

- More effective for search than the binary representation
- A tabular representation is simple (but very inefficient)

Vocabulary (Index Terms) |V|

Corpus |C|	a	abba	abe	ability	able	about	…	zooms
Doc_1	0	0	0	7	3	4	…	2
Doc_2	4	5	0	0	1	2	…	0
: : : :	:	: :	: :	: :	: :	: : :		: : :
Doc_n	6	0	1	3	0	1	…	0

Corpus: Document collection

15

© 2019, Jamie Callan

Two Lecture Outline

A quick introduction to…

- Ad-hoc retrieval
- Information needs & queries
- Document representation
- Exact match retrieval
  - Unranked Boolean
  - Ranked Boolean
- Indexes
  - Inverted lists
  - Term dictionary
- Document retrieval
  - TAAT
  - DAAT
- Query operators

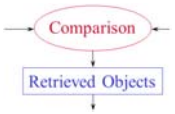
Goal: Provide an overview of search ("the Big Picture")

- Later lectures explore these topics in greater detail

16

© 2019, Jamie Callan

Retrieval Model #1: Unranked Boolean



Model: Retrieve documents iff they satisfy a Boolean expression

- **Examples:** "michelle AND obama", "clinton OR trump"
- The query specifies exact relevance criteria
  - Exact match retrieval
- The set of matching ("retrieved") documents is unordered
  - Often sorted by date

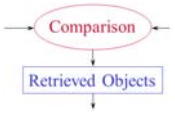
Query operators:

- AND, OR, ANDNOT, NEAR, DATE, …
- Typically these systems have rich query languages

17

© 2019, Jamie Callan

Retrieval Model #1: Unranked Boolean



This approach to document retrieval was invented first
…and was the dominant model until the early 1990s
…but it is no longer state-of-the-art

Why?

- Most people find it difficult to construct good Boolean queries
- Documents are returned in no particular order

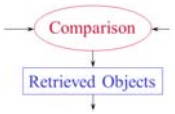
However, it is still used in many systems, and still important

- E.g., WestLaw, PubMed, first pass in Web search engines, …

18

© 2019, Jamie Callan

Retrieval Model #2: Ranked Boolean



Model: Retrieve documents iff they satisfy a Boolean expression

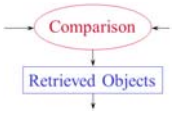
- Order the matching ("retrieved") documents by scores
- Document scores can be anything you want
  - But, typically Ranked Boolean systems have simple scores

Note: Unranked Boolean systems have implied document scores

- 1: Query matches document
- 0: Query doesn't match document

© 2019, Jamie Callan

Ranked Boolean: Calculating Scores



What is the score when a query term *t* occurs in document *d*?

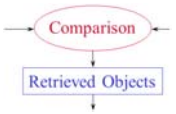
- **$tf_{t,d}$:** The frequency of term *t* in document *d*
- **$tf_{t,d} \times idf_t = tf_{t,d} \times \log (N / df_t)$:** Penalize common terms
  - N: Number of documents in the corpus
  - df_t: The number of documents that contain term *t*

Both types of weights are used

- **$tf_{t,d}$:** Invented first, easy to implement, only considers the document
- **$tf_{t,d} \times idf_t$:** More effective
  - Rewards terms that are frequent in this document, but not frequent in the corpus

© 2019, Jamie Callan

Ranked Boolean: Calculating Scores



Boolean queries have operators such as AND and OR

- cat AND mouse; john AND paul AND george AND ringo
- rich OR poor; obama OR bush OR clinton OR reagan

A prefix representation makes the query structure more apparent

- AND (cat mouse) AND (john paul george ringo)
- OR (rich poor) OR (obama bush clinton reagan)

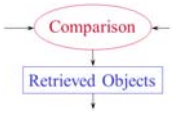
Notation: $q_{operator}\ (q_1 \ldots q_n)$

- q_{AND} (cat mouse) q_{AND} (john paul george ringo)
- q_{OR} (rich poor) q_{OR} (obama bush clinton reagan)

21

© 2019, Jamie Callan

Ranked Boolean: Calculating Scores



What is the score for AND operator $q_{AND}\ (q_1 \ldots q_n)$ on document j?

- score $(q_{AND}\ (q_1 \ldots q_n), d_j)$ = MIN (score (q_1, d_j), …, score (q_n, d_j))

What is the score for OR operator $q_{OR}\ (q_1 \ldots q_n)$ on document j?

- score $(q_{OR}\ (q_1 \ldots q_n), d_j)$ = MAX (score (q_1, d_j), …, score (q_n, d_j))
  - Consistent with the AND operator
- score $(q_{OR}\ (q_1 \ldots q_n), d_j)$ = MEAN (score (q_1, d_j), …, score (q_n, d_j))
  - Rewards documents that match many query terms
  - The semantics of an OR operator do not require this behavior
  - But, it is the behavior that people expect
  - Typically a little more effective than MAX

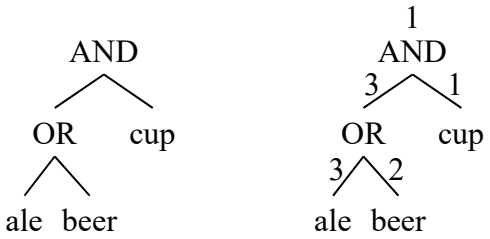
22

© 2019, Jamie Callan

Ranked Boolean: Calculating Scores

Document: … ale … cup … ale … beer … ale … beer

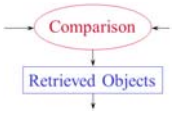
Query: (ale OR beer) AND cup



23

© 2019, Jamie Callan

Retrieval Model #2: Ranked Boolean



Advantages

- Very efficient
- Predictable, easy to explain, structured queries
- Works well enough when searchers know exactly what is wanted
- Results ordered by how redundantly a document satisfies a query
- Other term weighting methods can be used, too

Disadvantages

- It's still an Exact-Match model
- Usually it is difficult to get a good balance of Precision and Recall

24

© 2019, Jamie Callan

Exact-Match Retrieval: Unranked vs. Ranked Boolean Retrieval

Query: Trump AND Clinton

D_1	… Trump … Clinton … … Sanders …	**D_2**	Trump … Clinton… … Clinton … … Trump …	**D_3**	… Trump … Hillary … Bill … Sanders …

Three retrieval methods

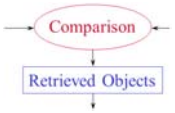
Unranked Boolean	Ranked Boolean	Best Match
D_1	D_2	D_2
D_2	D_1	D_1
(arbitrary order)		D_3

Which ranking is best?

- It depends on the task … sometimes unranked Boolean is enough

© 2019, Jamie Callan

Are Exact-Match Models Still Relevant?



Many people prefer exact-match Boolean models

- Professional searchers (e.g., librarians, paralegals)
- Some Web surfers (e.g., "Advanced Search" feature)
- What do they like? Control, predictability, understandability
- Preferred by 70% of WESTLAW searchers in a 2007 survey

  -- *James Allan, 2007*

Exact-match Boolean is a low-level part of Web search engines

- Massive corpus makes efficiency important
- Massive corpus makes partial matching less important

© 2019, Jamie Callan

Two Lecture Outline

A quick introduction to…

- Ad-hoc retrieval
- Information needs & queries
- Document representation
- Exact match retrieval
  - Unranked Boolean
  - Ranked Boolean
- Indexes
  - Inverted lists
- Document retrieval
  - TAAT
  - DAAT
- Query operators

Goal: Provide an overview of search ("the Big Picture")

- Later lectures explore these topics in greater detail

27

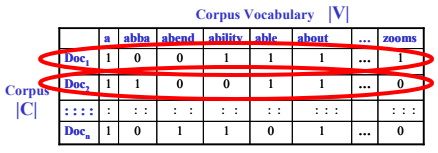
© 2019, Jamie Callan

Data Structures for Index Terms

Task: Evaluate the query "ability AND about"

One could compare to each document (row)

- Invented first
- Rows are bit vectors
- Complexity is O(|C|×|V|)
- Is this good?



Most terms are rare (occur in few documents)

- The vocabulary V is huge
- Nearly all documents fail to match the query
- Most of the O(|C|×|V|) effort is wasted effort

28

© 2019, Jamie Callan

Data Structures for Index Terms: Inverted Lists

Task: Evaluate the query "ability AND about"

One could compare to query terms (columns)

- Columns are bit vectors
- Columns are called inverted lists
- Complexity is O(|C|×|Q|)

Corpus Vocabulary

	a	abba	abend	ability	able	about	…	zooms
Corpus Doc_1	1	0	0	1	1	1	…	1
Doc_2	1	1	0	0	1	1	…	0
: : : :	:	: :	: :	: :	: :	: : :		: : :
Doc_n	1	0	1	1	0	1	…	0

Inverted Lists

Really important data structure!

Most terms are rare (occur in few documents)

- Nearly all documents fail to match the query
- More efficient

… but still, most of the O(|C|×|Q|) effort is wasted effort

© 2019, Jamie Callan

Are Fixed-Length Inverted Lists A Good Idea?

Early search engines used fixed-length inverted lists

- Simple
- Bit-vector operations are fast and easy to parallelize
- Very inefficient (1 bit or integer per document)

apple
1
0
0
1
:
0
1

Zipf's Law predicts that the median term occurs twice

- Rank of last term: Constant / Frequency = (0.1 × N) / 1
- Rank of median term: ½ rank of last term = ½ × (0.1 × N) / 1
- Frequency of median term: Constant / Rank of median term = 2

Thus, inverted lists almost always use sparse representations

© 2019, Jamie Callan

Sparse Representation of Inverted Lists

Simple approach: Store ids of documents that contain the word

- E.g., apple: length=18, docids: 1, 5, 6, 9, …
  - The term 'apple' occurs in 18 documents

A more typical notation for this course

- df_t=18, docids 1, 5, 6, 9, …

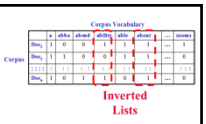
  df_t: document frequency (number of documents containing term t)

  docids: document identifiers

You must know this data structure & more advanced variants

© 2019, Jamie Callan

Different Types of Inverted Lists for the Term "Apple"



Different types of inverted lists support different capabilities

Binary Inverted lists

df:	4356
docid:	42
docid:	94
:	

Frequency Inverted lists

df:	4356
docid:	42
tf:	3
docid:	94
tf:	1
:	

Positional Inverted lists

df:	4356
docid:	42
tf:	3
locs:	14
	83
	157
docid:	94
tf:	1
locs:	65
:	

df: document frequency
docid: sequential document ids
tf: term frequency ($tf_{t,d}$)
locs: locations where t appears in d

Binary

- Unranked Boolean
- AND, OR, …

Frequency

- Ranking
- SUM, …

Positional

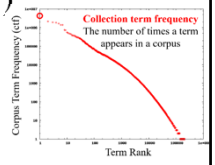
- NEAR/n, …

© 2019, Jamie Callan

Inverted Indexes

After indexing, there are many inverted lists

- One per term in the vocabulary (typically 10^6 to 10^8)
- Very skewed size distribution (Zipf's Law)
- Very skewed access patterns



An inverted index consists of two parts

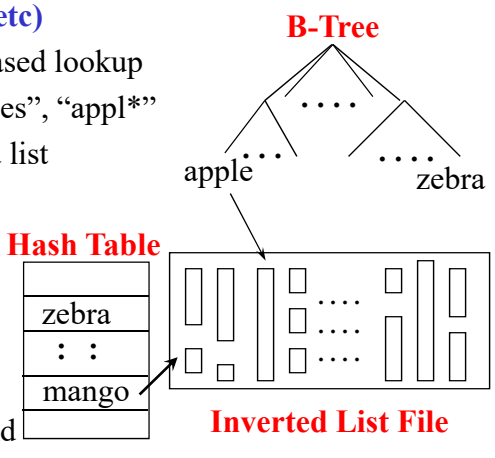
- Inverted file(s) that contain inverted lists
  - An object database containing the inverted lists
- An access mechanism
  - Term string → inverted list
  - Term id → inverted list
  - Sometimes combined with the term dictionary

© 2019, Jamie Callan

Inverted Indexes: Two Common Access Methods

How is a file of inverted lists accessed?

- **B-Tree (B+ Tree, B* Tree, etc)**
  - Exact-match and range-based lookup
    - » "apple", "apple – apples", "appl*"
  - *O*(log n) lookups to find a list
  - Usually easy to expand
- **Hash table**
  - Exact-match lookup
    - » "apple"
  - O(1) lookups to find a list
  - May be complex to expand



© 2019, Jamie Callan

Two Lecture Outline

A quick introduction to…

- Ad-hoc retrieval
- Information needs & queries
- Document representation
- Exact match retrieval
  - Unranked Boolean
  - Ranked Boolean
- Indexes
  - Inverted lists
- Document retrieval
  - TAAT
  - DAAT
- Query operators

Goal: Provide an overview of search ("the Big Picture")

- Later lectures explore these topics in greater detail

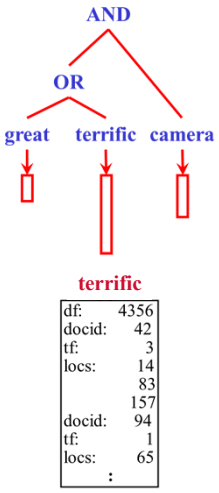
35

© 2019, Jamie Callan

Document Retrieval

There are three approaches to query processing

- Term-at-a-Time (TAAT)
- Document-at-a-Time (DAAT)
- TAAT / DAAT hybrids
  - Important in large-scale systems, but not covered in this class



36

© 2019, Jamie Callan

Document Retrieval: Term-at-a-Time (TAAT) Query Evaluation

Key ideas

- Fully process $list_i$ before proceeding to $list_{i+1}$
- Each time a list is processed, partial document scores are updated

© 2019, Jamie Callan

Document Retrieval: Term-at-a-Time (TAAT) Query Evaluation

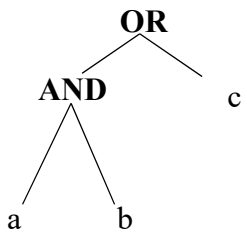
Query: (a AND b) OR c

© 2019, Jamie Callan

Document Retrieval: Term-at-a-Time (TAAT) Query Evaluation

Query: (a AND b) OR c

Parsed Query



39

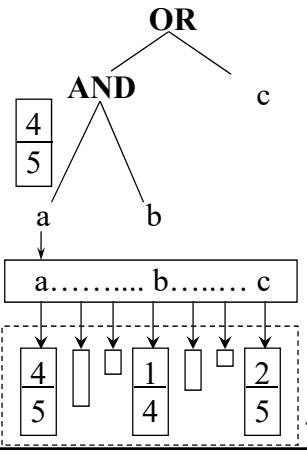
© 2019, Jamie Callan

Document Retrieval: Term-at-a-Time (TAAT) Query Evaluation

Query: (a AND b) OR c

1. Read inverted list for 'a' from inverted list database

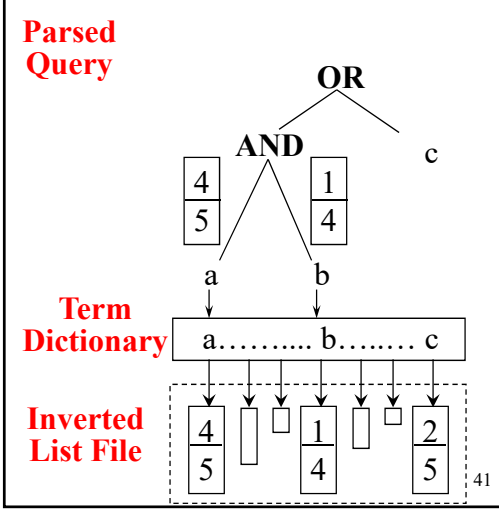
Parsed Query



40

© 2019, Jamie Callan

Document Retrieval: Term-at-a-Time (TAAT) Query Evaluation



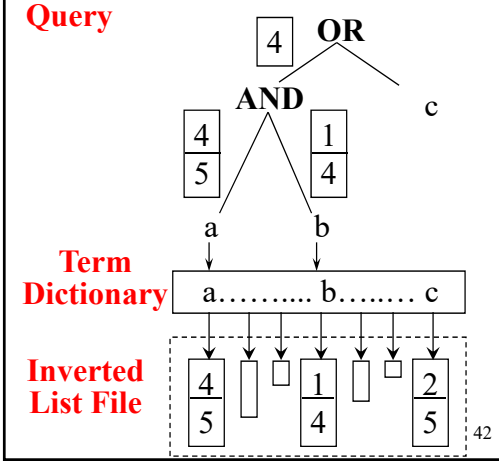
41

Query: (a AND b) OR c

1. Read inverted list for 'a' from inverted list database
2. Read inverted list for 'b' from inverted list database

© 2019, Jamie Callan

Document Retrieval: Term-at-a-Time (TAAT) Query Evaluation



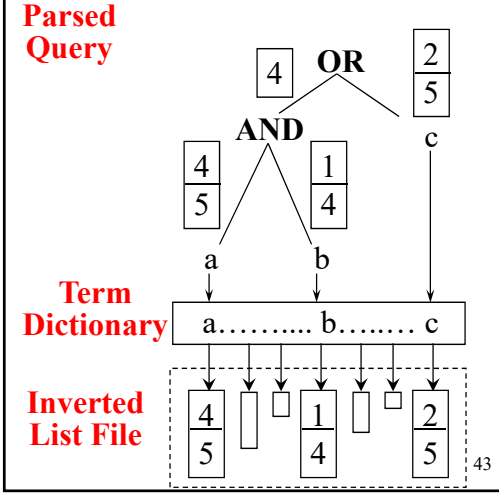
42

Query: (a AND b) OR c

1. Read inverted list for 'a' from inverted list database
2. Read inverted list for 'b' from inverted list database
3. AND operator: Intersect the inverted lists for 'a' and 'b'

© 2019, Jamie Callan

Document Retrieval: Term-at-a-Time (TAAT) Query Evaluation



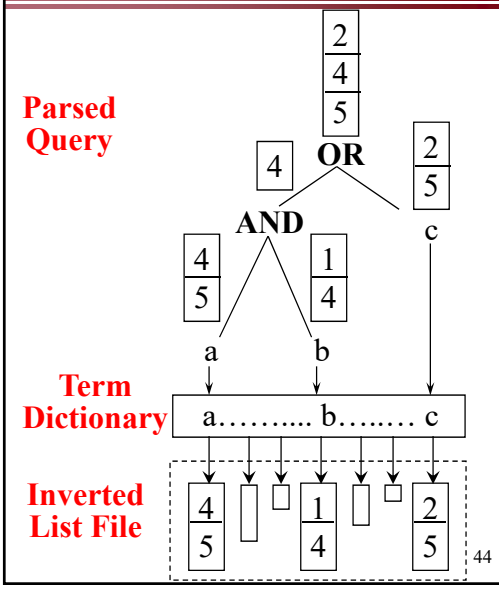
43

Query: (a AND b) OR c

1. Read inverted list for ‘a’ from inverted list database
2. Read inverted list for ‘b’ from inverted list database
3. AND operator: Intersect the inverted lists for ‘a’ and ‘b’
4. Read inverted list for ‘c’ from inverted list database

© 2019, Jamie Callan

Document Retrieval: Term-at-a-Time (TAAT) Query Evaluation



44

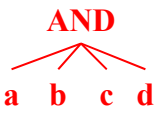
Query: (a AND b) OR c

1. Read inverted list for ‘a’ from inverted list database
2. Read inverted list for ‘b’ from inverted list database
3. AND operator: Intersect the inverted lists for ‘a’ and ‘b’
4. Read inverted list for ‘c’ from inverted list database
5. OR operator: Union of AND operator results and ‘c’ inverted list

© 2019, Jamie Callan

Document Retrieval: Term-at-a-Time (TAAT) Query Evaluation

Query: #AND (a b c d)



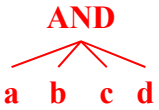
Evaluation strategy

- Retrieve a
- Retrieve b
- a AND b → $Result_{AND\_1}$
- Retrieve c
- $Result_{AND\_1}$ AND c → $Result_{AND\_2}$
- Retrieve d
- $Result_{AND\_2}$ AND d → $Result_{Q}$

© 2019, Jamie Callan

Document Retrieval: Term-at-a-Time (TAAT) Query Evaluation

Query: #AND (a b c d)



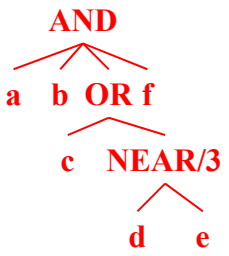
Characteristics

- Each query operator stores in RAM up to 3 lists simultaneously
  - arg_1, arg_2, result
- Peak query operatory memory usage for this query
  - 3 lists in RAM simultaneously
  - size (arg_1) + size (arg_2) + size (result) bytes

© 2019, Jamie Callan

Document Retrieval: Term-at-a-Time (TAAT) Query Evaluation

Query: #AND (a b #OR (c #NEAR/3 (d e)) f)



Peak memory usage (probably)

- 5 lists in memory simultaneously
- size (a AND b) +
  size (c) +
  size (d) + size (e) + size (d NEAR/3 e) bytes

© 2019, Jamie Callan

Document Retrieval: Term-at-a-Time (TAAT) Query Evaluation

Easy to understand and build

- Thus, we cover them first

Very efficient

- Little wasted effort
- This is more apparent when we consider DAAT

Memory usage is uncontrolled

- A query of depth d must store d+2 lists in RAM

Causes of memory problems

- Queries with frequent terms (long lists)
- Complex queries (more lists)
- Systems that process multiple queries in parallel (contention)

Rarely used in large systems

© 2019, Jamie Callan

Two Lecture Outline

A quick introduction to…

- Ad-hoc retrieval
- Information needs & queries
- Document representation
- Exact match retrieval
  - Unranked Boolean
  - Ranked Boolean
- Indexes
  - Inverted lists
  - Term dictionary
- Document retrieval
  - TAAT
  - DAAT
- Query operators

Goal: Provide an overview of search ("the Big Picture")

- Later lectures explore these topics in greater detail

© 2019, Jamie Callan