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
  - Term dictionary
- Query processing
  - TAAT
  - DAAT
- Query operators

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

- Later lectures explore these topics in greater detail

Term Dictionary

Main purpose

- Map term string to term id (integer)
- Example: "stocks" → 14,319

Often term dictionaries are stored in RAM

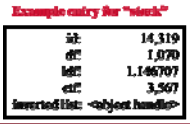
- So, they are a good place to store other frequently-access information, too
- E.g., statistics such as df, idf, ctf
- E.g., an inverted list pointer

Example entry for "stock"

id:	14,319
df:	1,070
idf:	1.146707
ctf:	3,567
inverted list:	<object handle>

40

Term Dictionary: Storage



Hash tables

- **O(1) lookup**
  - Very fast
- **Capabilities**
  - Exact-match lookup

B-Trees

- **O (log n) lookup**
  - Fast, but not O(1)
- **Capabilities**
  - Exact-match lookup
  - Range lookup
  - Prefix lookup

41

Term Dictionary: Storage

The gov2 web corpus (25 million pages) has 40 million terms

- The term complete dictionary might require 1.6 GB
- Most search engines won't devote that much RAM to a term dictionary

Suppose we want to use a tiered term dictionary

- Frequent terms in RAM (e.g., ctf ≥ 1,000)
- Less frequent terms on disk (e.g., ctf < 1,000)
- The Indri search engine uses this strategy

What percentage of terms are stored in RAM?

Term Dictionary: Storage

What percentage of terms have ctf < 1,000?

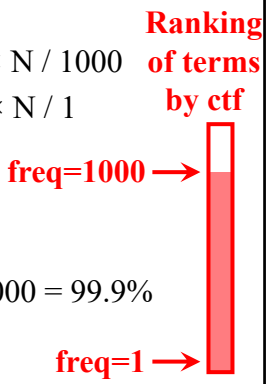
- **Zipf's Law:** $\text{Rank}_t \times \text{Frequency}_t = A \times N$
- **Rank of a word that occurs 1,000 times:** $A \times N / 1000$
- **Rank of a word that occurs 1 time:** $A \times N / 1$
  - An estimate of the vocabulary size
- **The percentage of terms with ctf < 1,000:**

  $(\text{Rank}_{Max} - \text{Rank}_{1000}) / \text{Rank}_{Max} =$

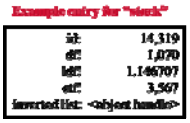
  $((A \times N) - (A \times N / 1000)) / (A \times N) = 999 / 1000 = 99.9\%$

But … Zipf's Law is inaccurate at the tails

- How does this affect the prediction?



Term Dictionary: Storage



How good is the prediction on the gov2 corpus?

- 25 million .gov web pages with a vocabulary of 40 million terms

How many terms occur more than 1,000 times?

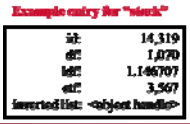
Predicted: 40,000 frequent terms (0.10%)

Actual: 185,000 frequent terms (0.46%)

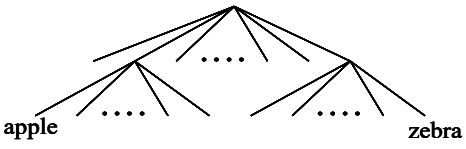
About 99.5% of terms have ctf ≤ 1,000

- 99.9% vs. 99.5%

Term Dictionary: How Does Indri Do It?

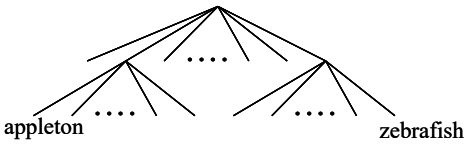


Frequent Vocabulary (ctf ≥ 1000)



In Memory

Infrequent Vocabulary (ctf < 1000)



Memory and Disk

For the Gov2 collection

- About 185,000 frequent terms in RAM (00.5%)
- About 39.78 million infrequent terms on disk (99.5%)

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
- Query processing
  - TAAT
  - DAAT
- Query operators

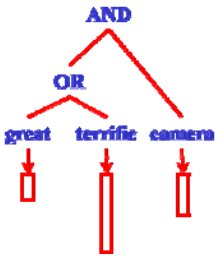
Goal: Provide an overview of search ("the Big Picture")

- Later lectures explore these topics in greater detail

Query Processing

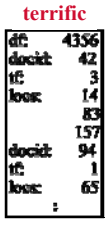
Query processing is a simple task

- **Input:** Two or more lists (e.g., inverted lists)
  - One list for each query argument
- **Output:** One list
  - The result list



There are three approaches to query processing

- Term-at-a-Time (TAAT)
- Document-at-a-Time (DAAT)
- TAAT / DAAT hybrids



Query Processing: Term-at-a-Time Query Evaluation

Key idea

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

48

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

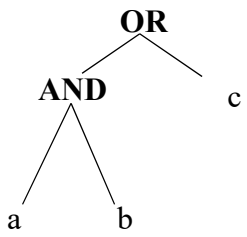
Query: (a AND b) OR c

49

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

Query: (a AND b) OR c

Parsed Query



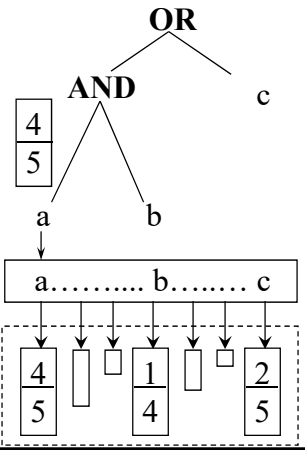
50

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

Query: (a AND b) OR c

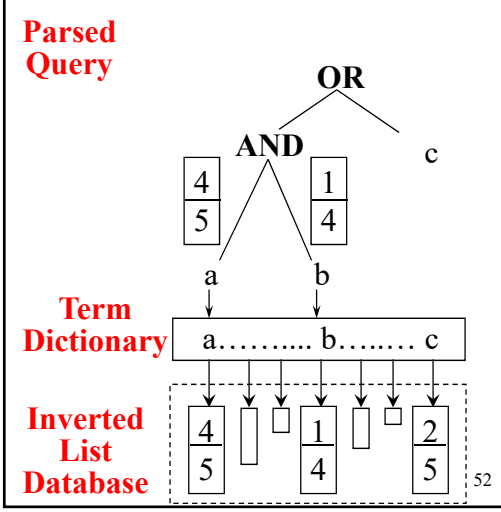
1. Read inverted list for 'a' from inverted list database

Parsed Query



51

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

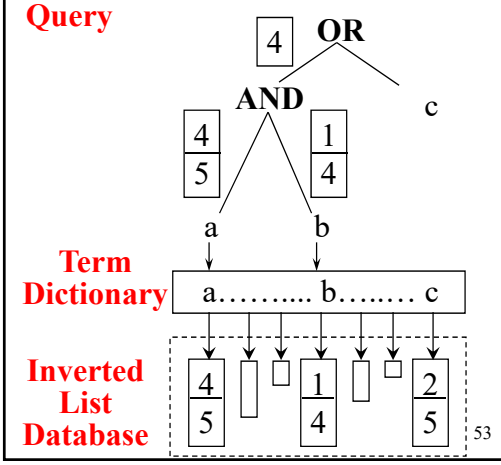


52

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

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

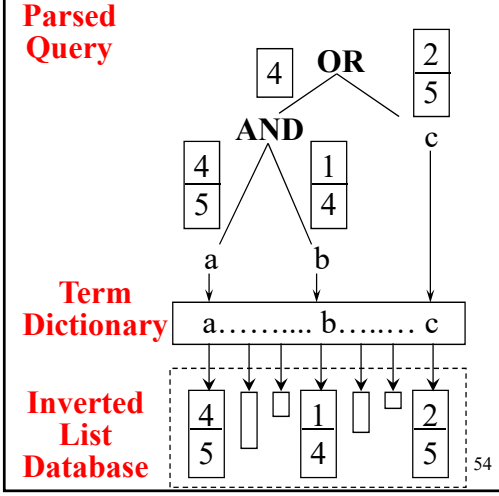


53

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'

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

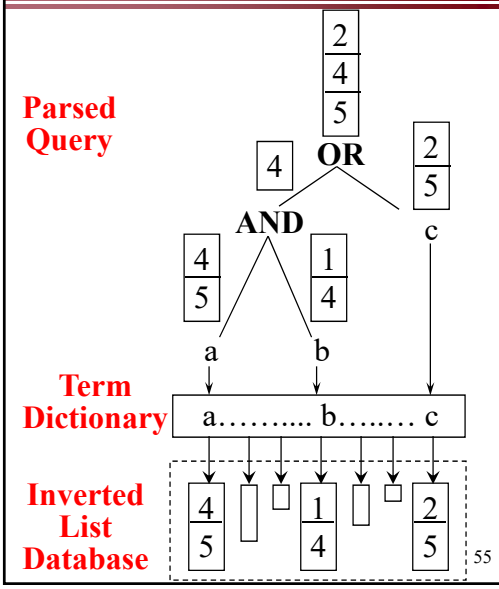


54

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

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



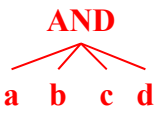
55

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

Query Processing: Term-at-a-Time Query Evaluation

Query: #AND (a b c d)

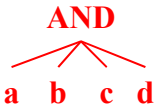


Evaluation strategy

- 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 → $Result_{Q}$

Query Processing: Term-at-a-Time Query Evaluation

Query: #AND (a b c d)

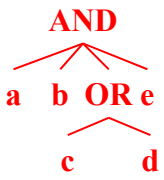


Characteristics

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

Query Processing: Term-at-a-Time Query Evaluation

Query: #AND (a b #OR (c d) e)



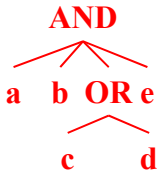
Evaluation strategy

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

58

Query Processing: Term-at-a-Time Query Evaluation

Query: #AND (a b #OR (c d) e)



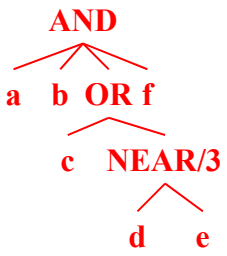
Peak memory usage (probably)

- 4 lists in memory simultaneously
- size (a AND b) +
  size (c) + size (d) + size (c OR d) bytes

59

Query Processing: Term-at-a-Time Query Evaluation

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

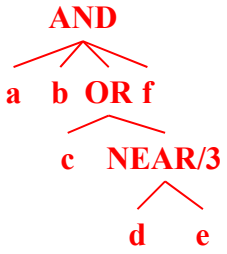


Evaluation strategy

- Retrieve a
- Retrieve b
- a AND b → $Result_{AND\_1}$
- Retrieve c
- Retrieve d
- Retrieve e
- d NEAR/3 e → $Result_{NEAR}$
- c OR $Result_{NEAR}$ → $Result_{OR}$
- $Result_{AND\_1}$ AND $Result_{OR}$ → $Result_{AND\_2}$
- Retrieve f
- $Result_{AND\_2}$ AND f → $Result_{Q}$

Query Processing: Term-at-a-Time 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

Query Processing: Term-at-a-Time Query Evaluation

TAAT systems are easy to build

- Each operator is a series of f (arg_1, arg_2) → result operations
- Conceptually, easy to understand
- Thus, we cover them first

TAAT systems are very efficient

- Little wasted effort
  - This isn't apparent now, but will be when we consider DAAT

Query Processing: Term-at-a-Time Query Evaluation

TAAT memory usage is uncontrolled

- Each query operator stores in RAM up to 3 lists simultaneously
  - arg_1, arg_2, result
- A query of depth d must store d+2 lists in RAM simultaneously

TAAT systems can run out of memory

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

Rarely used in large-scale systems

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
- Query processing
  - TAAT
  - DAAT
- Query operators

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

- Later lectures explore these topics in greater detail

Query Processing: Document-at-a-Time Query Evaluation

Key idea

- Compute a complete score for doc_i before proceeding to doc_{i+1}

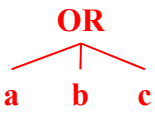
The following example assumes an unranked Boolean model.

- All scores are 1
- The same architecture can be used for ranked Boolean

Query Processing: Document-at-a-Time Query Evaluation

Starting condition:

- The query is #OR (a b c)
- Retrieve the inverted list for each term



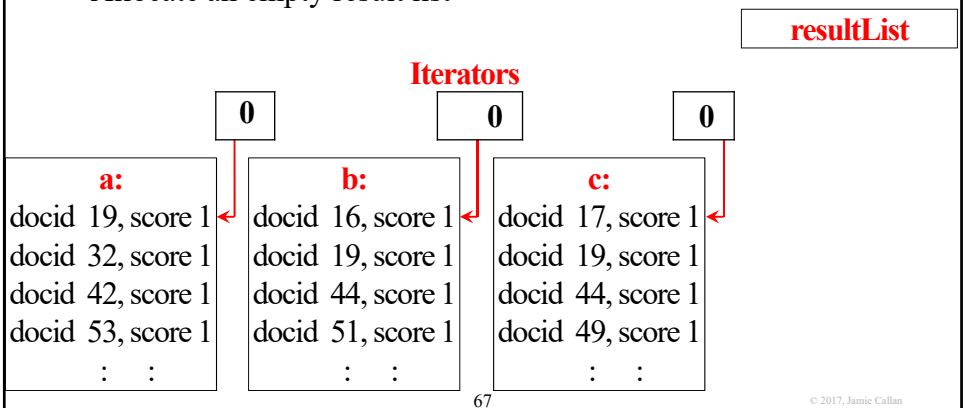
a:	b:	c:
docid 19, score 1	docid 16, score 1	docid 17, score 1
docid 32, score 1	docid 19, score 1	docid 19, score 1
docid 42, score 1	docid 44, score 1	docid 44, score 1
docid 53, score 1	docid 51, score 1	docid 49, score 1
: :	: :	: :

66

Query Processing: Document-at-a-Time Query Evaluation

Initialization:

- Allocate iterators for processing inverted lists
- Allocate an empty result list

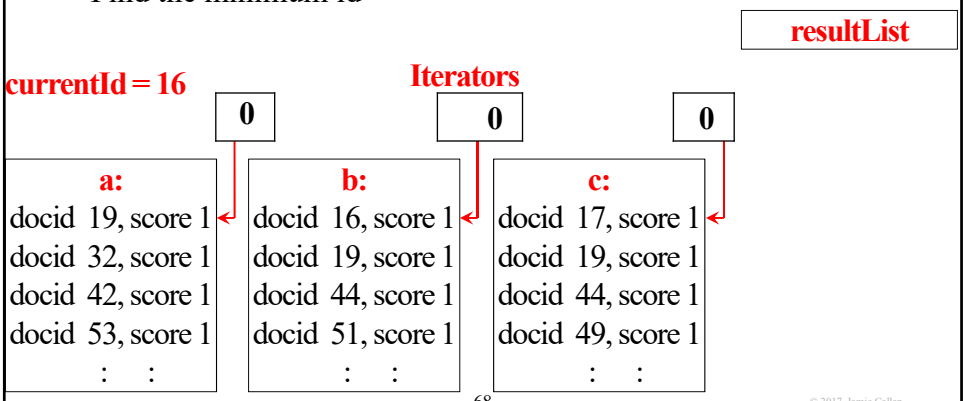


67

Query Processing: Document-at-a-Time Query Evaluation

Beginning of loop

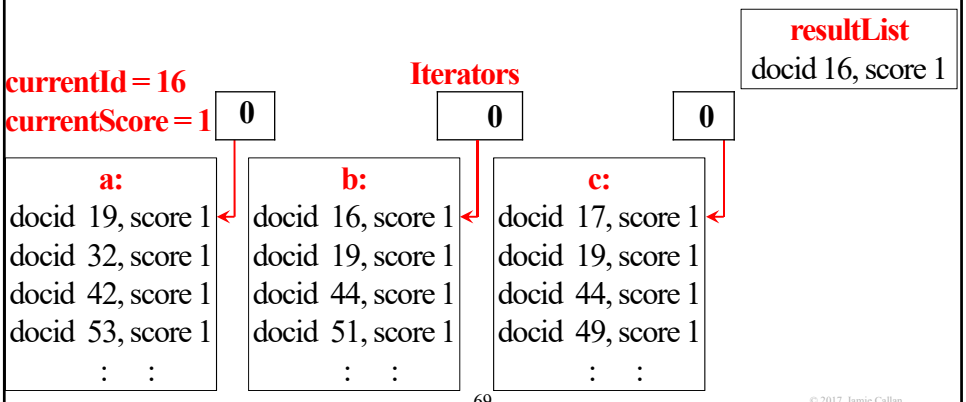
- Examine the current document id in each list
- Find the minimum id



68

Query Processing: Document-at-a-Time Query Evaluation

- Examine each list that contains currentId to compute currentScore
- Store the result

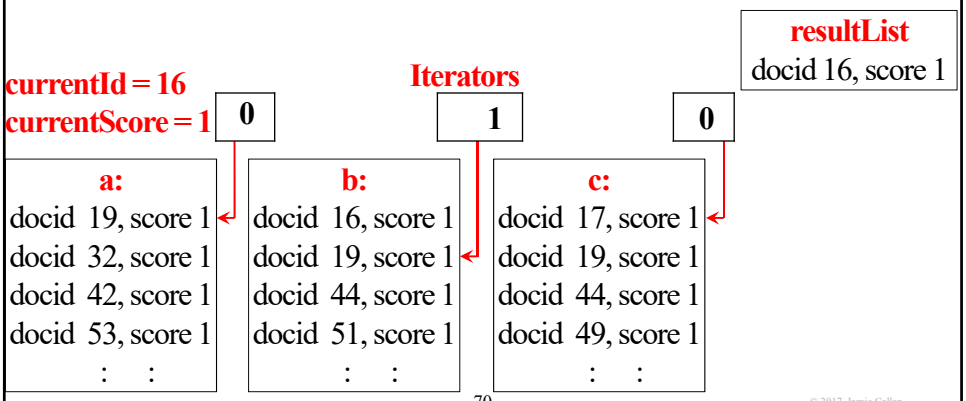


69

Query Processing: Document-at-a-Time Query Evaluation

- Advance each pointer that points to the currentId

End of loop

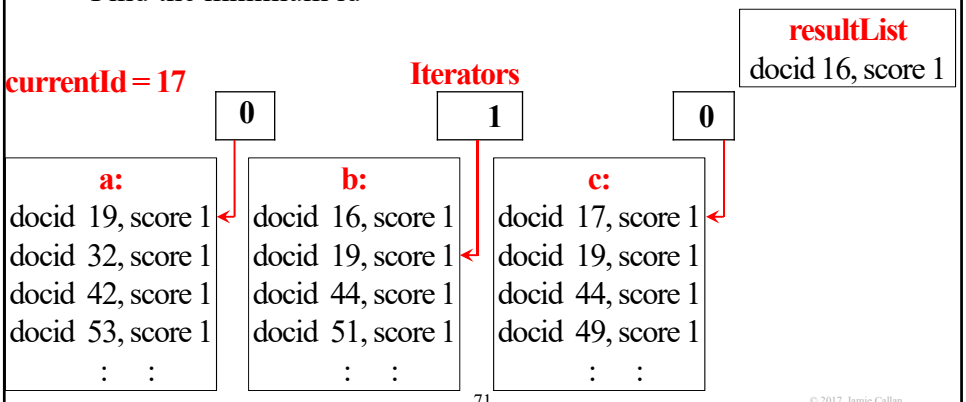


70

Query Processing: Document-at-a-Time Query Evaluation

Beginning of loop

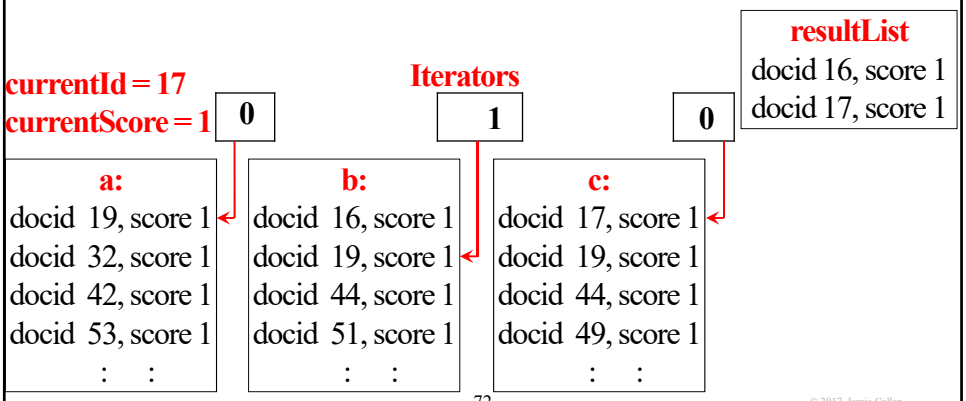
- Examine the current document id in each list
- Find the minimum id



71

Query Processing: Document-at-a-Time Query Evaluation

- Examine each list that contains currentId to compute currentScore
- Store the result

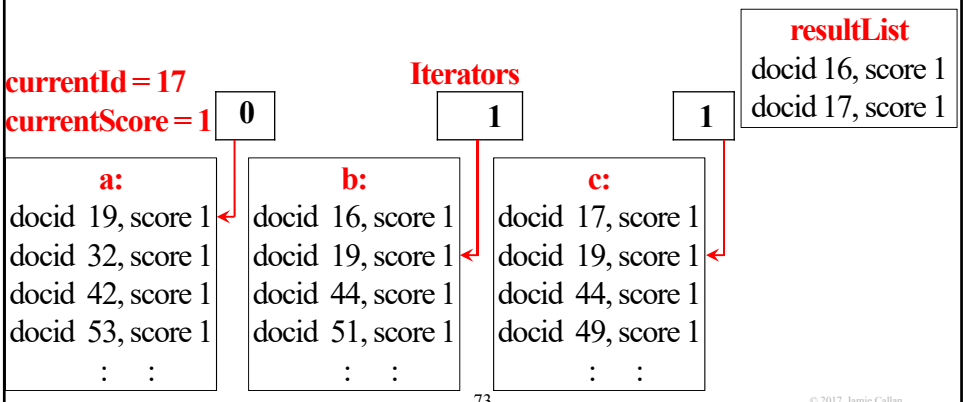


72

Query Processing: Document-at-a-Time Query Evaluation

- Advance each pointer that points to the currentId

End of loop

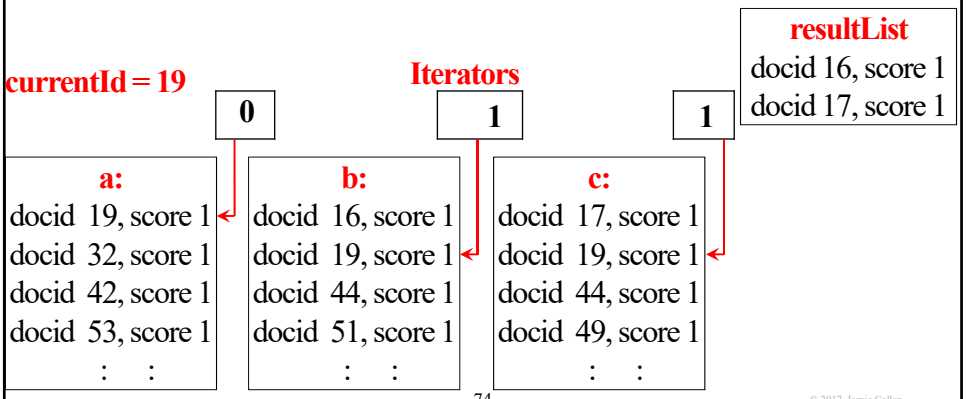


73

Query Processing: Document-at-a-Time Query Evaluation

Beginning of loop

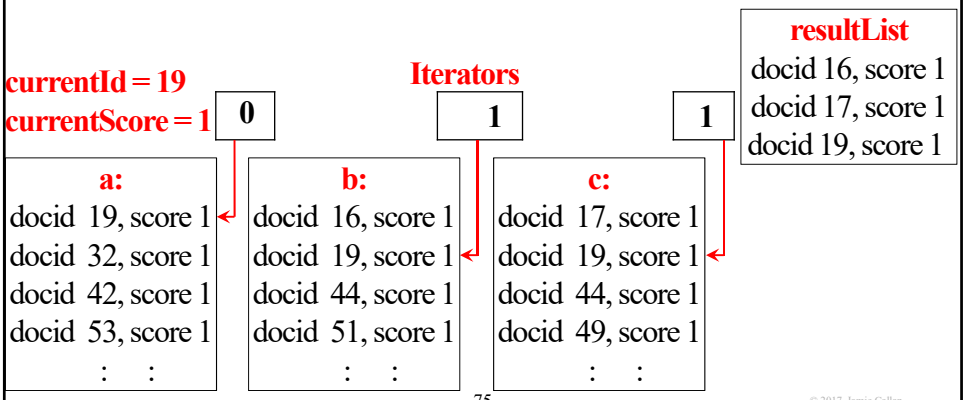
- Examine the current document id in each list
- Find the minimum id



74

Query Processing: Document-at-a-Time Query Evaluation

- Examine each list that contains currentId to compute currentScore
- Store the result

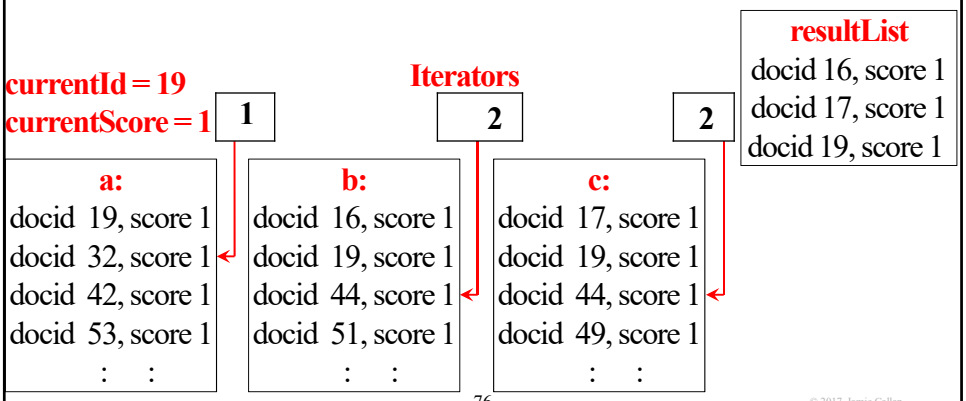


75

Query Processing: Document-at-a-Time Query Evaluation

- Advance each pointer that points to the currentId

End of loop

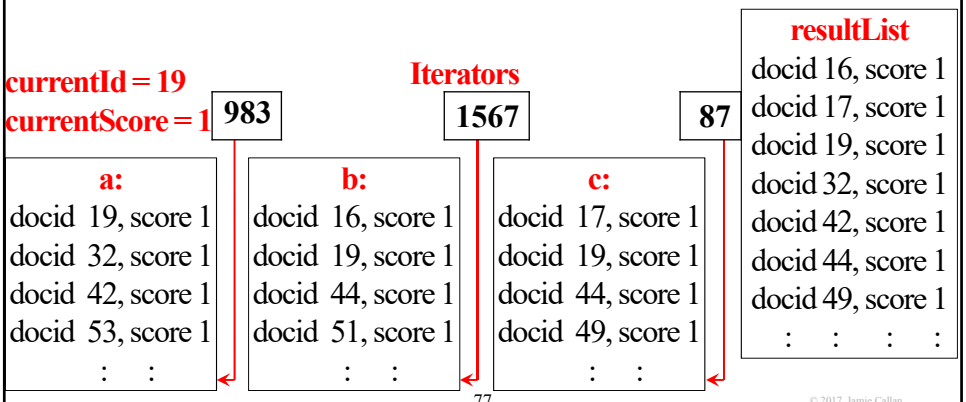


76

Query Processing: Document-at-a-Time Query Evaluation

Continue the loop until every inverted list is fully processed

Return the resultList



77

DAAT Query Processing: Implementation Notes

The simple implementation requires nested loops

- E.g., to find the minimum document id
- E.g., to compute the score for the current document id
- E.g., to advance pointers

A more efficient implementation combines loops

- If this list has the current docid
  - Update the score
  - Advance the pointer

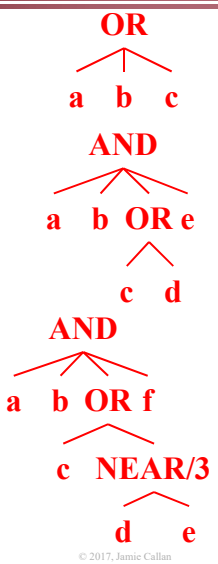
There are many opportunities for clever optimization

Query Processing: Document-at-a-Time Query Evaluation

How does DAAT support structured queries?

- Conceptually, it is something like this

```
q.initialize ()
while ( q.hasNext () )
  q.evalNext () returns next [docid, score] tuple
```



Each call to q.evalNext() traverses the entire tree

- This is a little inefficient … but not horrible
- The tricky part is figuring out the next docid
- Many opportunities for optimization

DAAT Query Processing Characteristics

DAAT memory usage is easy to control

- It needs simultaneous access to all inverted lists (which seems bad)
- But … inverted lists are read from disk into RAM in blocks
  - E.g., read the inverted list in 64MB blocks
- When the end of the current block is reached, read the next block
- The block size determines how much RAM the query uses

Many query evaluation optimizations are possible

- E.g., only partial evaluation of documents with low scores

Frequently used in large-scale systems

Query Processing: TAAT / DAAT Hybrids

Hybrid TAAT and DAAT architectures are common

- To get a blend of efficiency and memory control
- E.g., block-based TAAT
  - Compute TAAT over blocks of document ids
- A popular research topic

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
- Query processing
  - TAAT
  - DAAT
- Query operators

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

- Later lectures explore these topics in greater detail

Query Operators

Usually exact match systems have rich query languages

- Ranking is weak or missing
- Query languages provide control over what is matched

Today's focus

- Proximity operators
  - NEAR/n and WINDOW/n
- Types (classes) of query operators

The goal is to prepare you for HW1

Proximity Operators: The NEAR Operator

NEAR/n: Distance between adjacent arguments is ≥ 0 && ≤ n terms

- Query: "President NEAR/2 Obama"

  Document texts:

"President Obama"	**Matches (distance is 1)**
"President Barack Obama"	**Matches (distance is 2)**
"President Barack H. Obama"	**Doesn't match (distance is 3)**
"Obama is President"	**Doesn't match (distance is -2)**

Sentence/n: Like NEAR/n, but distance is measured in sentences

Paragraph/n: Like NEAR/n, but distance is measured in paragraphs

84

Proximity Operators: The NEAR Operator

The NEAR/n operator is used to match names and phrases

- Arguments must be matched in order
- n specifies the maximum distance between adjacent terms

Examples

- #NEAR/1 (barack obama)
  - Matches "barack obama"
  - Doesn't match "barack hussein obama" or "obama, barack"
- #NEAR/3 (barack obama)
  - Matches "barack obama" and "barack hussein obama"
- #NEAR/4 (a b c) matches (a x x b x x x c)

85

Proximity Operators: A Simple Greedy NEAR/n Algorithm

Typically proximity operators have complexity O(|C|)

- A single pass down each inverted list
- Similar in complexity to merging sorted lists

Proximity Operators: A Simple Greedy NEAR/n Algorithm

a		b	
df:	47	df:	95
doc:	19	doc:	23
tf:	1	tf:	1
locs:	7	locs:	99
doc:	27	doc:	27
tf:	3	tf:	4
locs:	47	locs:	48
	98		49
	132		133
doc:	92		134
…		doc:	148
		...	

Query: #NEAR/3 (a b)

Proximity Operators: A Simple Greedy NEAR/n Algorithm

a		b	
df:	47	df:	95
doc:	19	doc:	23
tf:	1	tf:	1
locs:	7	locs:	99
doc:	27	doc:	27
tf:	3	tf:	4
locs:	47	locs:	48
	98		49
	132		133
doc:	92		134
…		doc:	148
		…	

Query: #NEAR/3 (a b)

Initialize doc iterators

88

Proximity Operators: A Simple Greedy NEAR/n Algorithm

a		b	
df:	47	df:	95
doc:	19	doc:	23
tf:	1	tf:	1
locs:	7	locs:	99
doc:	27	doc:	27
tf:	3	tf:	4
locs:	47	locs:	48
	98		49
	132		133
doc:	92		134
…		doc:	148
		…	

Query: #NEAR/3 (a b)

19 ≠ 23 (different doc)
Advance ptr of smallest doc

89

Proximity Operators: A Simple Greedy NEAR/n Algorithm

a		b	
df:	47	df:	95
doc:	19	doc:	23
tf:	1	tf:	1
locs:	7	locs:	99
doc:	27	doc:	27
tf:	3	tf:	4
locs:	47	locs:	48
	98		49
	132		133
doc:	92		134
…		doc:	148
		…	

Query: #NEAR/3 (a b)

90

Proximity Operators: A Simple Greedy NEAR/n Algorithm

a		b	
df:	47	df:	95
doc:	19	doc:	23
tf:	1	tf:	1
locs:	7	locs:	99
doc:	27	doc:	27
tf:	3	tf:	4
locs:	47	locs:	48
	98		49
	132		133
doc:	92		134
…		doc:	148
		…	

Query: #NEAR/3 (a b)

27 ≠ 23 (different doc)
Advance ptr of smallest doc

91

Proximity Operators: A Simple Greedy NEAR/n Algorithm

a		b	
df:	47	df:	95
doc:	19	doc:	23
tf:	1	tf:	1
locs:	7	locs:	99
doc:	27	doc:	27
tf:	3	tf:	4
locs:	47	locs:	48
	98		49
	132		133
doc:	92		134
…		doc:	148
		…	

Query: #NEAR/3 (a b)

92

Proximity Operators: A Simple Greedy NEAR/n Algorithm

a		b	
df:	47	df:	95
doc:	19	doc:	23
tf:	1	tf:	1
locs:	7	locs:	99
doc:	27	doc:	27
tf:	3	tf:	4
locs:	47	locs:	48
	98		49
	132		133
doc:	92		134
…		doc:	148
		…	

Query: #NEAR/3 (a b)

Same document
Initialize location ptrs

93

Proximity Operators: A Simple Greedy NEAR/n Algorithm

a		b	
df:	47	df:	95
doc:	19	doc:	23
tf:	1	tf:	1
locs:	7	locs:	99
doc:	27	doc:	27
tf:	3	tf:	4
locs:	47	locs:	48
	98		49
	132		133
doc:	92		134
…		doc:	148
		…	

Query: #NEAR/3 (a b)

48 – 47 ≤ n (match)
Record match
Advance all loc ptrs

94

Proximity Operators: A Simple Greedy NEAR/n Algorithm

a		b	
df:	47	df:	95
doc:	19	doc:	23
tf:	1	tf:	1
locs:	7	locs:	99
doc:	27	doc:	27
tf:	3	tf:	4
locs:	47	locs:	48
	98		49
	132		133
doc:	92		134
…		doc:	148
		…	

Query: #NEAR/3 (a b)

95

Proximity Operators: A Simple Greedy NEAR/n Algorithm

a		b	
df:	47	df:	95
doc:	19	doc:	23
tf:	1	tf:	1
locs:	7	locs:	99
doc:	27	doc:	27
tf:	3	tf:	4
locs:	47	locs:	48
	98		49
	132		133
doc:	92		134
…		doc:	148
		…	

Query: #NEAR/3 (a b)

98 > 49 (no match)
Advance ptr of smallest loc

96

Proximity Operators: A Simple Greedy NEAR/n Algorithm

a		b	
df:	47	df:	95
doc:	19	doc:	23
tf:	1	tf:	1
locs:	7	locs:	99
doc:	27	doc:	27
tf:	3	tf:	4
locs:	47	locs:	48
	98		49
	132		133
doc:	92		134
…		doc:	148
		…	

Query: #NEAR/3 (a b)

97

Proximity Operators: A Simple Greedy NEAR/n Algorithm

a		b	
df:	47	df:	95
doc:	19	doc:	23
tf:	1	tf:	1
locs:	7	locs:	99
doc:	27	doc:	27
tf:	3	tf:	4
locs:	47	locs:	48
	98		49
	132		133
doc:	92		134
…		doc:	148
		…	

Query: #NEAR/3 (a b)

133 – 98 > n (no match)
Advance ptr of smallest loc

Proximity Operators: A Simple Greedy NEAR/n Algorithm

a		b	
df:	47	df:	95
doc:	19	doc:	23
tf:	1	tf:	1
locs:	7	locs:	99
doc:	27	doc:	27
tf:	3	tf:	4
locs:	47	locs:	48
	98		49
	132		133
doc:	92		134
…		doc:	148
		…	

Query: #NEAR/3 (a b)

Proximity Operators: A Simple Greedy NEAR/n Algorithm

a		b	
df:	47	df:	95
doc:	19	doc:	23
tf:	1	tf:	1
locs:	7	locs:	99
doc:	27	doc:	27
tf:	3	tf:	4
locs:	47	locs:	48
	98		49
	132		133
doc:	92		134
…		doc:	148
		…	

Query: #NEAR/3 (a b)

133 – 132 ≤ n (match)
Record match
Advance all loc ptrs

a's locs are exhausted.
No more matches are possible in this document
Advance all doc ptrs.

100

Proximity Operators: A Simple Greedy NEAR/n Algorithm

a		b	
df:	47	df:	95
doc:	19	doc:	23
tf:	1	tf:	1
locs:	7	locs:	99
doc:	27	doc:	27
tf:	3	tf:	4
locs:	47	locs:	48
	98		49
	132		133
doc:	92		134
…		doc:	148
		…	

Query: #NEAR/3 (a b)

Continue until the inverted lists are exhausted

101

Proximity Operators: A Simple Greedy NEAR/n Algorithm

a		b		c	
df:	47	df:	95	df:	14
doc:	19	doc:	23	doc:	23
tf:	1	tf:	1	tf:	1
locs:	7	locs:	99	loc:	99
doc:	27	doc:	27	doc:	27
tf:	3	tf:	4	tf:	4
locs:	47	locs:	48	locs:	46
	98		49		51
	132		133		114
doc:	92		134		137
…		doc:	148	doc:	129
		…		…	

Query: #NEAR/3 (a b c)

102

Proximity Operators: A Simple Greedy NEAR/n Algorithm

a		b		c	
df:	47	df:	95	df:	14
doc:	19	doc:	23	doc:	23
tf:	1	tf:	1	tf:	1
locs:	7	locs:	99	loc:	99
doc:	27	doc:	27	doc:	27
tf:	3	tf:	4	tf:	4
locs:	47	locs:	48	locs:	46
	98		49		51
	132		133		114
doc:	92		134		137
…		doc:	148	doc:	129
		…		…	

Query: #NEAR/3 (a b c)

Initialize doc iterators

103

Proximity Operators: A Simple Greedy NEAR/n Algorithm

a		b		c	
df:	47	df:	95	df:	14
doc:	19	doc:	23	doc:	23
tf:	1	tf:	1	tf:	1
locs:	7	locs:	99	loc:	99
doc:	27	doc:	27	doc:	27
tf:	3	tf:	4	tf:	4
locs:	47	locs:	48	locs:	46
	98		49		51
	132		133		114
doc:	92		134		137
…		doc:	148	doc:	129
		…		…	

Query: #NEAR/3 (a b c)

19 ≠ 23 (different doc)
Advance ptr of smallest doc

104

Proximity Operators: A Simple Greedy NEAR/n Algorithm

a		b		c	
df:	47	df:	95	df:	14
doc:	19	doc:	23	doc:	23
tf:	1	tf:	1	tf:	1
locs:	7	locs:	99	loc:	99
doc:	27	doc:	27	doc:	27
tf:	3	tf:	4	tf:	4
locs:	47	locs:	48	locs:	46
	98		49		51
	132		133		114
doc:	92		134		137
…		doc:	148	doc:	129
		…		…	

Query: #NEAR/3 (a b c)

105

Proximity Operators: A Simple Greedy NEAR/n Algorithm

a		b		c	
df:	47	df:	95	df:	14
doc:	19	doc:	23	doc:	23
tf:	1	tf:	1	tf:	1
locs:	7	locs:	99	loc:	99
doc:	27	doc:	27	doc:	27
tf:	3	tf:	4	tf:	4
locs:	47	locs:	48	locs:	46
	98		49		51
	132		133		114
doc:	92		134		137
…		doc:	148	doc:	129
		…		…	

Query: #NEAR/3 (a b c)

27 ≠ 23 (different doc)
Advance ptrs of smallest docs

106

Proximity Operators: A Simple Greedy NEAR/n Algorithm

a		b		c	
df:	47	df:	95	df:	14
doc:	19	doc:	23	doc:	23
tf:	1	tf:	1	tf:	1
locs:	7	locs:	99	loc:	99
doc:	27	doc:	27	doc:	27
tf:	3	tf:	4	tf:	4
locs:	47	locs:	48	locs:	46
	98		49		51
	132		133		114
doc:	92		134		137
…		doc:	148	doc:	129
		…		…	

Query: #NEAR/3 (a b c)

107

Proximity Operators: A Simple Greedy NEAR/n Algorithm

a		b		c	
df:	47	df:	95	df:	14
doc:	19	doc:	23	doc:	23
tf:	1	tf:	1	tf:	1
locs:	7	locs:	99	loc:	99
doc:	27	doc:	27	doc:	27
tf:	3	tf:	4	tf:	4
locs:	47	locs:	48	locs:	46
	98		49		51
	132		133		114
doc:	92		134		137
…		doc:	148	doc:	129
		…		…	

Query: #NEAR/3 (a b c)

Same document
Initialize location ptrs

108

Proximity Operators: A Simple Greedy NEAR/n Algorithm

a		b		c	
df:	47	df:	95	df:	14
doc:	19	doc:	23	doc:	23
tf:	1	tf:	1	tf:	1
locs:	7	locs:	99	loc:	99
doc:	27	doc:	27	doc:	27
tf:	3	tf:	4	tf:	4
locs:	47	locs:	48	locs:	46
	98		49		51
	132		133		114
doc:	92		134		137
…		doc:	148	doc:	129
		…		…	

Query: #NEAR/3 (a b c)

48 – 47 ≤ n (match)
46 – 48 < 0 (no match)
Advance ptr of smallest loc

109

Proximity Operators: A Simple Greedy NEAR/n Algorithm

a		b		c	
df:	47	df:	95	df:	14
doc:	19	doc:	23	doc:	23
tf:	1	tf:	1	tf:	1
locs:	7	locs:	99	loc:	99
doc:	27	doc:	27	doc:	27
tf:	3	tf:	4	tf:	4
locs:	47	locs:	48	locs:	46
	98		49		51
	132		133		114
doc:	92		134		137
…		doc:	148	doc:	129
		…		…	

Query: #NEAR/3 (a b c)

110

Proximity Operators: A Simple Greedy NEAR/n Algorithm

a		b		c	
df:	47	df:	95	df:	14
doc:	19	doc:	23	doc:	23
tf:	1	tf:	1	tf:	1
locs:	7	locs:	99	loc:	99
doc:	27	doc:	27	doc:	27
tf:	3	tf:	4	tf:	4
locs:	47	locs:	48	locs:	46
	98		49		51
	132		133		114
doc:	92		134		137
…		doc:	148	doc:	129
		…		…	

Query: #NEAR/3 (a b c)

48 – 47 ≤ n (match)
51 – 48 ≤ n (match)
Record match
Advance all loc ptrs

111

Proximity Operators: A Simple Greedy NEAR/n Algorithm

a		b		c	
df:	47	df:	95	df:	14
doc:	19	doc:	23	doc:	23
tf:	1	tf:	1	tf:	1
locs:	7	locs:	99	loc:	99
doc:	27	doc:	27	doc:	27
tf:	3	tf:	4	tf:	4
locs:	47	locs:	48	locs:	46
	98		49		51
	132		133		114
doc:	92		134		137
…		doc:	148	doc:	129
		…		…	

Query: #NEAR/3 (a b c)

112

Proximity Operators: A Simple Greedy NEAR/n Algorithm

a		b		c	
df:	47	df:	95	df:	14
doc:	19	doc:	23	doc:	23
tf:	1	tf:	1	tf:	1
locs:	7	locs:	99	loc:	99
doc:	27	doc:	27	doc:	27
tf:	3	tf:	4	tf:	4
locs:	47	locs:	48	locs:	46
	98		49		51
	132		133		114
doc:	92		134		137
…		doc:	148	doc:	129
		…		…	

Query: #NEAR/3 (a b c)

49 – 98 < 0 (no match)
Advance ptr of smallest loc

113

Proximity Operators: A Simple Greedy NEAR/n Algorithm

a		b		c	
df:	47	df:	95	df:	14
doc:	19	doc:	23	doc:	23
tf:	1	tf:	1	tf:	1
locs:	7	locs:	99	loc:	99
doc:	27	doc:	27	doc:	27
tf:	3	tf:	4	tf:	4
locs:	47	locs:	48	locs:	46
	98		49		51
	132		133		114
doc:	92		134		137
…		doc:	148	doc:	129
		…		…	

Query: #NEAR/3 (a b c)

Proximity Operators: A Simple Greedy NEAR/n Algorithm

a		b		c	
df:	47	df:	95	df:	14
doc:	19	doc:	23	doc:	23
tf:	1	tf:	1	tf:	1
locs:	7	locs:	99	loc:	99
doc:	27	doc:	27	doc:	27
tf:	3	tf:	4	tf:	4
locs:	47	locs:	48	locs:	46
	98		49		51
	132		133		114
doc:	92		134		137
…		doc:	148	doc:	129
		…		…	

Query: #NEAR/3 (a b c)

133 – 98 > n (no match)
Advance ptr of smallest loc

Proximity Operators: A Simple Greedy NEAR/n Algorithm

a		b		c	
df:	47	df:	95	df:	14
doc:	19	doc:	23	doc:	23
tf:	1	tf:	1	tf:	1
locs:	7	locs:	99	loc:	99
doc:	27	doc:	27	doc:	27
tf:	3	tf:	4	tf:	4
locs:	47	locs:	48	locs:	46
	98		49		51
	132		133		114
doc:	92		134		137
…		doc:	148	doc:	129
		…		…	

Query: #NEAR/3 (a b c)

Proximity Operators: A Simple Greedy NEAR/n Algorithm

a		b		c	
df:	47	df:	95	df:	14
doc:	19	doc:	23	doc:	23
tf:	1	tf:	1	tf:	1
locs:	7	locs:	99	loc:	99
doc:	27	doc:	27	doc:	27
tf:	3	tf:	4	tf:	4
locs:	47	locs:	48	locs:	46
	98		49		51
	132		133		114
doc:	92		134		137
…		doc:	148	doc:	129
		…		…	

Query: #NEAR/3 (a b c)

133 – 132 > n (match)
114 – 133 < 0 (no match)
Advance ptr of smallest loc

Proximity Operators: A Simple Greedy NEAR/n Algorithm

a		b		c	
df:	47	df:	95	df:	14
doc:	19	doc:	23	doc:	23
tf:	1	tf:	1	tf:	1
locs:	7	locs:	99	loc:	99
doc:	27	doc:	27	doc:	27
tf:	3	tf:	4	tf:	4
locs:	47	locs:	48	locs:	46
	98		49		51
	132		133		114
doc:	92		134		137
…		doc:	148	doc:	129
		…		…	

Query: #NEAR/3 (a b c)

118

Proximity Operators: A Simple Greedy NEAR/n Algorithm

a		b		c	
df:	47	df:	95	df:	14
doc:	19	doc:	23	doc:	23
tf:	1	tf:	1	tf:	1
locs:	7	locs:	99	loc:	99
doc:	27	doc:	27	doc:	27
tf:	3	tf:	4	tf:	4
locs:	47	locs:	48	locs:	46
	98		49		51
	132		133		114
doc:	92		134		137
…		doc:	148	doc:	129
		…		…	

Query: #NEAR/3 (a b c)

133 – 132 ≤ n (match)
137 – 133 > n (no match)
Advance ptr of smallest loc

a's locs are exhausted.
No more matches are possible in this document
Advance all doc ptrs

119

Proximity Operators: A Simple Greedy NEAR/n Algorithm

a		b		c	
df:	47	df:	95	df:	14
doc:	19	doc:	23	doc:	23
tf:	1	tf:	1	tf:	1
locs:	7	locs:	99	loc:	99
doc:	27	doc:	27	doc:	27
tf:	3	tf:	4	tf:	4
locs:	47	locs:	48	locs:	46
	98		49		51
	132		133		114
doc:	92		134		137
…		doc:	148	doc:	129
		…		…	

Query: #NEAR/3 (a b c)

Perhaps you expected b's loc ptr to be advanced

This is a flaw in the simple greedy algorithm. The smallest loc ptr isn't always the optimal loc ptr

Better algorithms are possible, but also more complex

120

Proximity Operators: A Simple Greedy NEAR/n Algorithm

a		b		c	
df:	47	df:	95	df:	14
doc:	19	doc:	23	doc:	23
tf:	1	tf:	1	tf:	1
locs:	7	locs:	99	loc:	99
doc:	27	doc:	27	doc:	27
tf:	3	tf:	4	tf:	4
locs:	47	locs:	48	locs:	46
	98		49		51
	132		133		114
doc:	92		134		137
…		doc:	148	doc:	129
		…		…	

Query: #NEAR/3 (a b c)

Continue until the inverted lists are exhausted

121

Proximity Operators: NEAR/n FAQ

Query: #NEAR/2 (a b)

Text: **a b** x **a** x x x **a** x x **b** x x **a** x **b**

- There are two matches {1, 2} and {14, 16}
- Results for the NEAR operator: tf =2, and locations=2, 16

Query: #NEAR/2 (a b c)

Text: a a b b c c

- There are two matches {1, 3, 5} and {2, 4, 6}
- Results for the NEAR operator: tf =2, and locations=5, 6

Proximity Operators: NEAR/n FAQ

Query: #NEAR/3 (a b)

Text: **a b** c **b**

- There is one match {1, 2}
  - A query term can match only one text term
- Results for the NEAR operator: tf =1, and locations=2

Query: #NEAR/3 (a b)

Text: **b a** c **a**

- There are no matches
  - The order of NEAR query arguments is important

Proximity Operators: NEAR/n FAQ

Query: #NEAR/2 (a a b)

Text: a b a a b

- The greedy algorithm does not handle duplicate arguments
- The distance constraint can be modified to support this case
  - E.g., $0 < \text{distance}(t_1, t_2) \leq n$
  - Maybe also a better algorithm for advancing pointers

These cases haven't been studied much by researchers

- But Google seems to support "apple apple pie"

Proximity Operators: NEAR/n FAQ

Query: #NEAR/3 (a b c)

Text: **a b** d **b** x x **c**

- The greedy algorithm fails to find a match
  - It considers {1, 2, 7} (the first location for each term)
  - {1, 2, 7} fails to match, so the smallest location pointer advances
    - » a
  - The list for a is exhausted, so this text does not match
- A better algorithm would find a match at {1, 4, 7}
  - More accurate, but also slower
  - The greedy algorithm is usually sufficient in practice
- Use the greedy algorithm for your homework

Proximity Operators: The Window (or Unordered Window) Operator

The WINDOW/n operator is used to match related concepts

- Arguments can be in any order
- n specifies the maximum distance between any pair of terms

Examples

- WINDOW/100 (obama merkel putin)
  - We don't care which order they occur in

Typically proximity operators have complexity O(|C|)

- A single pass down each inverted list

126

Proximity Operators: The Window (or Unordered Window) Operator

a		b	
df:	47	df:	95
doc:	19	doc:	23
tf:	1	tf:	1
locs:	7	locs:	99
doc:	27	doc:	27
tf:	3	tf:	4
locs:	47	locs:	48
	98		49
	132		133
doc:	92		134
…		doc:	148
		...	

Query: # WINDOW/20 (a b)

127

Proximity Operators: The Window (or Unordered Window) Operator

a		b	
df:	47	df:	95
doc:	19	doc:	23
tf:	1	tf:	1
locs:	7	locs:	99
doc:	27	doc:	27
tf:	3	tf:	4
locs:	47	locs:	48
	98		49
	132		133
doc:	92		134
…		doc:	148
		…	

Query: # WINDOW/20 (a b)

Initialize doc iterators

128

Proximity Operators: The Window (or Unordered Window) Operator

a		b	
df:	47	df:	95
doc:	19	doc:	23
tf:	1	tf:	1
locs:	7	locs:	99
doc:	27	doc:	27
tf:	3	tf:	4
locs:	47	locs:	48
	98		49
	132		133
doc:	92		134
…		doc:	148
		…	

Query: # WINDOW/20 (a b)

19 ≠ 23 (different doc)
Advance ptr of smallest doc

129

Proximity Operators: The Window (or Unordered Window) Operator

a		b	
df:	47	df:	95
doc:	19	doc:	23
tf:	1	tf:	1
locs:	7	locs:	99
doc:	27	doc:	27
tf:	3	tf:	4
locs:	47	locs:	48
	98		49
	132		133
doc:	92		134
…		doc:	148
		…	

Query: # WINDOW/20 (a b)

130

Proximity Operators: The Window (or Unordered Window) Operator

a		b	
df:	47	df:	95
doc:	19	doc:	23
tf:	1	tf:	1
locs:	7	locs:	99
doc:	27	doc:	27
tf:	3	tf:	4
locs:	47	locs:	48
	98		49
	132		133
doc:	92		134
…		doc:	148
		…	

Query: # WINDOW/20 (a b)

27 ≠ 23 (different doc)
Advance ptr of smallest doc

131

Proximity Operators: The Window (or Unordered Window) Operator

a		b	
df:	47	df:	95
doc:	19	doc:	23
tf:	1	tf:	1
locs:	7	locs:	99
doc:	27	doc:	27
tf:	3	tf:	4
locs:	47	locs:	48
	98		49
	132		133
doc:	92		134
…		doc:	148
		…	

Query: # WINDOW/20 (a b)

132

Proximity Operators: The Window (or Unordered Window) Operator

a		b	
df:	47	df:	95
doc:	19	doc:	23
tf:	1	tf:	1
locs:	7	locs:	99
doc:	27	doc:	27
tf:	3	tf:	4
locs:	47	locs:	48
	98		49
	132		133
doc:	92		134
…		doc:	148
		…	

Query: # WINDOW/20 (a b)

Same document
Initialize location ptrs

133

Proximity Operators: The Window (or Unordered Window) Operator

a		b	
df:	47	df:	95
doc:	19	doc:	23
tf:	1	tf:	1
locs:	7	locs:	99
doc:	27	doc:	27
tf:	3	tf:	4
locs:	47	locs:	48
	98		49
	132		133
doc:	92		134
…		doc:	148
		…	

Query: # WINDOW/20 (a b)

|48 – 47| < 20 (match)
Record match
Advance all loc ptrs

134

Proximity Operators: The Window (or Unordered Window) Operator

a		b	
df:	47	df:	95
doc:	19	doc:	23
tf:	1	tf:	1
locs:	7	locs:	99
doc:	27	doc:	27
tf:	3	tf:	4
locs:	47	locs:	48
	98		49
	132		133
doc:	92		134
…		doc:	148
		…	

Query: # WINDOW/20 (a b)

135

Proximity Operators: The Window (or Unordered Window) Operator

a		b	
df:	47	df:	95
doc:	19	doc:	23
tf:	1	tf:	1
locs:	7	locs:	99
doc:	27	doc:	27
tf:	3	tf:	4
locs:	47	locs:	48
	98		49
	132		133
doc:	92		134
…		doc:	148
		…	

Query: # WINDOW/20 (a b)

|98 – 49| ≥ 20 (no match)
Advance ptr of smallest loc

136

Proximity Operators: The Window (or Unordered Window) Operator

a		b	
df:	47	df:	95
doc:	19	doc:	23
tf:	1	tf:	1
locs:	7	locs:	99
doc:	27	doc:	27
tf:	3	tf:	4
locs:	47	locs:	48
	98		49
	132		133
doc:	92		134
…		doc:	148
		…	

Query: # WINDOW/20 (a b)

137

Proximity Operators: The Window (or Unordered Window) Operator

a		b	
df:	47	df:	95
doc:	19	doc:	23
tf:	1	tf:	1
locs:	7	locs:	99
doc:	27	doc:	27
tf:	3	tf:	4
locs:	47	locs:	48
	98		49
	132		133
doc:	92		134
…		doc:	148
		…	

Query: # WINDOW/20 (a b)

|133 – 98| ≥ 20 (no match)
Advance ptr of smallest loc

Proximity Operators: The Window (or Unordered Window) Operator

a		b	
df:	47	df:	95
doc:	19	doc:	23
tf:	1	tf:	1
locs:	7	locs:	99
doc:	27	doc:	27
tf:	3	tf:	4
locs:	47	locs:	48
	98		49
	132		133
doc:	92		134
…		doc:	148
		…	

Query: # WINDOW/20 (a b)

Proximity Operators: The Window (or Unordered Window) Operator

a		b	
df:	47	df:	95
doc:	19	doc:	23
tf:	1	tf:	1
locs:	7	locs:	99
doc:	27	doc:	27
tf:	3	tf:	4
locs:	47	locs:	48
	98		49
	132		133
doc:	92		134
…		doc:	148
		…	

Query: # WINDOW/20 (a b)

|133 – 132| < 20 (match)
Record match
Advance all loc ptrs

a's locs are exhausted.
No more matches are possible in this document
Advance all doc ptrs.

Proximity Operators: The Window (or Unordered Window) Operator

a		b	
df:	47	df:	95
doc:	19	doc:	23
tf:	1	tf:	1
locs:	7	locs:	99
doc:	27	doc:	27
tf:	3	tf:	4
locs:	47	locs:	48
	98		49
	132		133
doc:	92		134
…		doc:	148
		…	

Query: # WINDOW/20 (a b)

Continue until the inverted lists are exhausted

Proximity Operators: The Window (or Unordered Window) Operator

Implementation note

- A document term can only match the query once
- **Query:** #WINDOW/100 (obama merkel putin)
- **Document**: obama … merkel … putin … merkel … obama
- There is just one match here

One can imagine other implementations, but this is the norm

- Usually more complicated matching doesn't improve accuracy

Two Important Data Structures

Inverted lists

'apple"

df:	27
docid:	14
tf:	2
locs:	37
	92
docid:	89
	:
	:

You already know about inverted lists (stored on disk)

Score lists

'apple'

docid	score
14	3
89	2
127	2
:	:
:	:

#AND and #OR query operators produce these (in memory)

An unranked system has score = 1

How do we get from inverted lists to score lists?

Three Types of Query Operators

There are three important classes of query operators

- **Operators that combine inverted lists**
  - E.g., #NEAR/n, #SYNONYM, #WINDOW/n, …
  - Dynamically create new index terms / concepts
- **Operators that convert inverted lists to scores**
  - Usually invisible or implied
  - Usually just one – we will call it #SCORE
- **Operators that combine score lists**
  - E.g., #AND, #OR, #AVERAGE, #MAX, …
  - Combine evidence

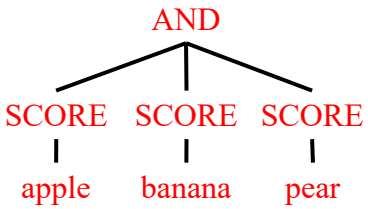
Every (?) query operator belongs to one of these classes

invertedList
df
docid tf loc+
docid tf loc+
: : :

scoreList
df
docid score
docid score
...

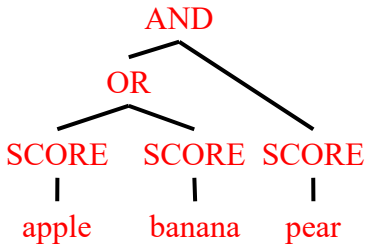
Three Types of Query Operators

#AND (apple banana pear)



Three Types of Query Operators

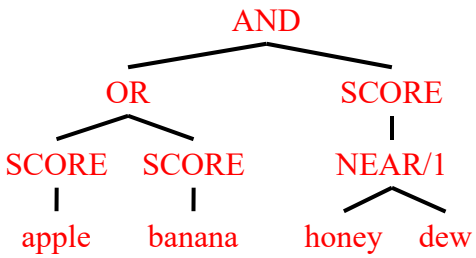
#AND (#OR (apple banana) pear)



146

Three Types of Query Operators

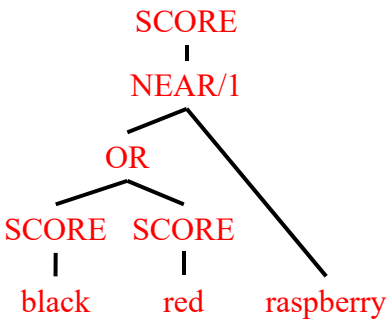
#AND (#OR (apple banana) #NEAR/1 (honey dew))



147

Three Types of Query Operators

Some queries are not allowed
#SCORE (#NEAR/1 (#OR (black red) raspberry))



NEAR/1 operates on inverted lists
It creates an inverted list for a new concept

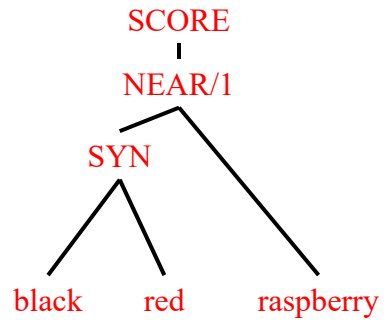
OR operates on score lists
It combines evidence

Use a SYNONYM operator to combine inverted lists

148

Three Types of Query Operators

This query is allowed
#SCORE (#NEAR/1 (#SYN (black red) raspberry))



NEAR/1 operates on inverted lists
It creates an inverted list for a new concept

149

OR vs SYN (SYNONYM)

OR and SYN look similar, but they are very different

SYN constructs new concepts dynamically

- By merging inverted lists

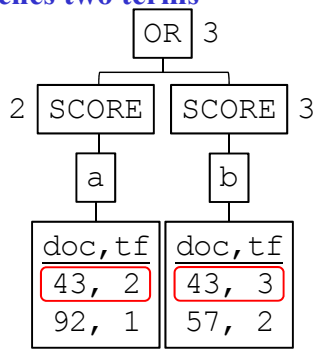
OR combines evidence about how well the document satisfies the information need

- Evidence obtained from matching multiple terms

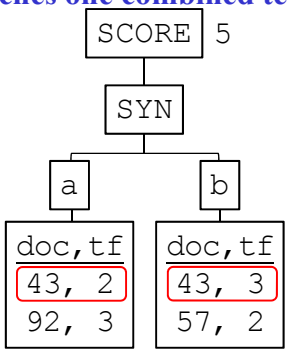
These are very different things

OR vs SYN (SYNONYM): Unranked Boolean Retrieval Model

#OR (a b)
Matches two terms



#SYN (a b)
Matches one combined term

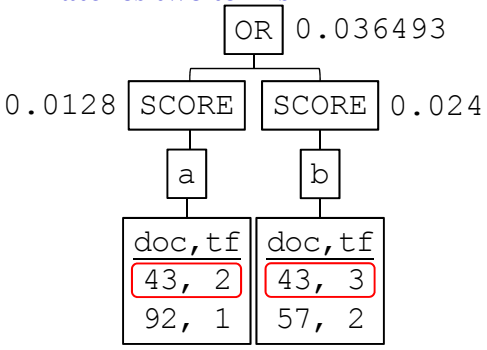


SCORE: tf

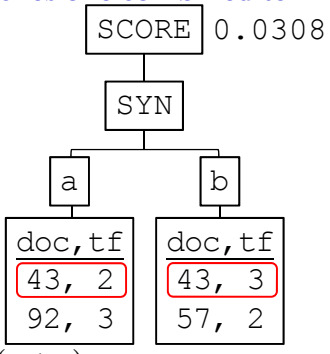
OR: MAX

OR vs SYN (SYNONYM): Indri Retrieval Model

#OR (a b)
Matches two terms



#SYN (a b)
Matches one combined term



SCORE: $p_{score}(q_i \mid d) = \dfrac{tf_{q_i,d} + \mu\, p_{MLE}(q_i \mid C)}{length(d) + \mu}$

OR: $p_{or}(q \mid d) = 1 - \prod_{q_i \in q} (1 - p(q_i \mid d))$

OR vs SYN (SYNONYM)

SYN behaves the same for all retrieval models

- i.e., merge inverted lists to create new concepts

Retrieval models treat the concepts produced by SYN differently

- **Unranked Boolean:** Prefers concepts that are frequent in this document
  - tf-oriented
- **Indri:** Prefers concepts that are frequent in this document, but rare across the corpus
  - tf.idf-oriented

Do Real Systems Have a SCORE Operator?

People don't write SCORE operators in their queries
… so how do SCORE operators get into queries?

The query parser can insert them automatically when needed

– If a query operator combines scores &&
its i^{th} query argument returns an inverted list
Then wrap the i^{th} argument in a SCORE operator

– E.g., #AND (a b) → #AND (#SCORE (a) #SCORE (b))

– The sample QryEval software does this

Outline

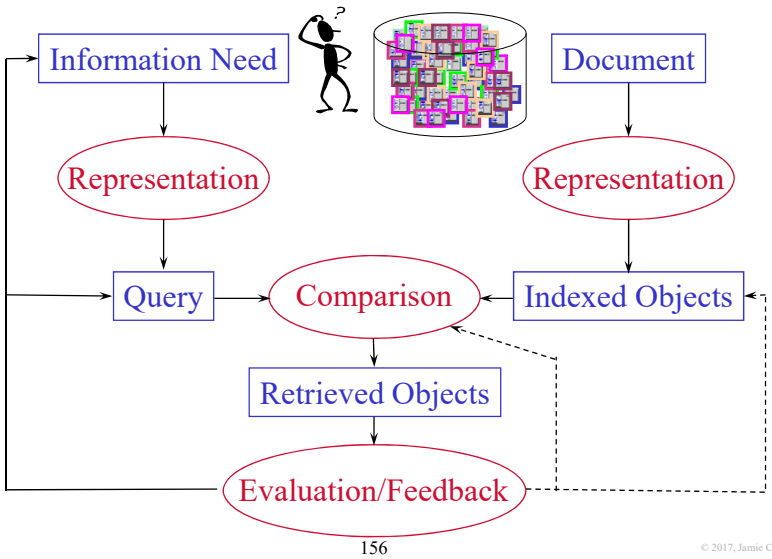
Introduction to…

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

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

- Later lectures explore these topics in greater detail

Overview of Information Retrieval Processes



156

Waitlist Reminder

If you are on the waitlist…

- I will admit some people from the waitlist today or tomorrow
- Be sure to sign the attendance sheet to show that you are here

157