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
- Document retrieval
  - TAAT
  - DAAT
- Query operators

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

- Later lectures explore these topics in greater detail

Document Retrieval: Document-at-a-Time (DAAT) 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

Document Retrieval: Document-at-a-Time (DAAT) Query Evaluation

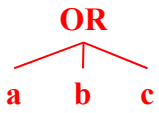
Starting condition:

- The query is #OR (a b c)

Document Retrieval: Document-at-a-Time (DAAT) Query Evaluation

Starting condition:

- The query is #OR (a b c)
- Parse the query

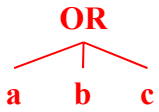


5

Document Retrieval: Document-at-a-Time (DAAT) Query Evaluation

Starting condition:

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



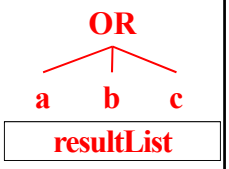
a:	b:	c:
docid 19	docid 16	docid 17
docid 32	docid 19	docid 19
docid 42	docid 44	docid 44
docid 53	docid 51	docid 49
: :	: :	: :

6

Document Retrieval: Document-at-a-Time (DAAT) Query Evaluation

Initialization:

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



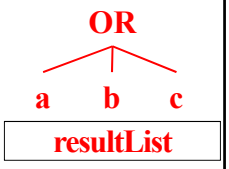
a:	b:	c:
docid 19	docid 16	docid 17
docid 32	docid 19	docid 19
docid 42	docid 44	docid 44
docid 53	docid 51	docid 49
: :	: :	: :

7

Document Retrieval: Document-at-a-Time (DAAT) Query Evaluation

Beginning of loop

- Examine the visible document id in each list
- Set currentId to the minimum id



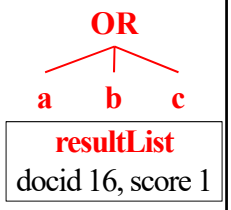
currentId = 16

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

8

Document Retrieval: Document-at-a-Time (DAAT) Query Evaluation

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



currentId = 16
currentScore = 1

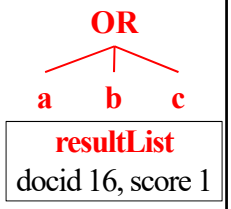
a:	b:	c:
docid 19	docid 16	docid 17
docid 32	docid 19	docid 19
docid 42	docid 44	docid 44
docid 53	docid 51	docid 49
: :	: :	: :

9

Document Retrieval: Document-at-a-Time (DAAT) Query Evaluation

- Advance each iterator that points to the currentId

End of loop



currentId = 16
currentScore = 1

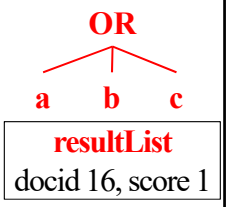
a:	b:	c:
docid 19	docid 16	docid 17
docid 32	docid 19	docid 19
docid 42	docid 44	docid 44
docid 53	docid 51	docid 49
: :	: :	: :

10

Document Retrieval: Document-at-a-Time (DAAT) Query Evaluation

Beginning of loop

- Examine the visible document id in each list
- Set currentId to the minimum id



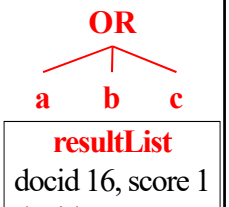
currentId = 17

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

11

Document Retrieval: Document-at-a-Time (DAAT) Query Evaluation

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



currentId = 17
currentScore = 1

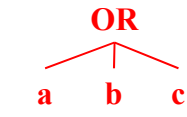
a:	b:	c:
docid 19	docid 16	docid 17
docid 32	docid 19	docid 19
docid 42	docid 44	docid 44
docid 53	docid 51	docid 49
: :	: :	: :

12

Document Retrieval: Document-at-a-Time (DAAT) Query Evaluation

- Advance each iterator that points to the currentId

End of loop



resultList
docid 16, score 1
docid 17, score 1

currentId = 17
currentScore = 1

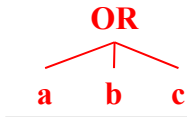
a:	b:	c:
docid 19	docid 16	docid 17
docid 32	docid 19	docid 19
docid 42	docid 44	docid 44
docid 53	docid 51	docid 49
: :	: :	: :

13

Document Retrieval: Document-at-a-Time (DAAT) Query Evaluation

Beginning of loop

- Examine the visible document id in each list
- Set currentId to the minimum id



resultList
docid 16, score 1
docid 17, score 1

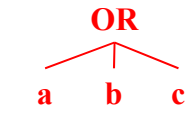
currentId = 19

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

14

Document Retrieval: Document-at-a-Time (DAAT) Query Evaluation

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



resultList
docid 16, score 1
docid 17, score 1
docid 19, score 1

currentId = 19
currentScore = 1

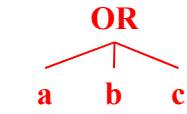
a:	b:	c:
docid 19	docid 16	docid 17
docid 32	docid 19	docid 19
docid 42	docid 44	docid 44
docid 53	docid 51	docid 49
: :	: :	: :

15

Document Retrieval: Document-at-a-Time (DAAT) Query Evaluation

- Advance each iterator that points to the currentId

End of loop



resultList
docid 16, score 1
docid 17, score 1
docid 19, score 1

currentId = 19
currentScore = 1

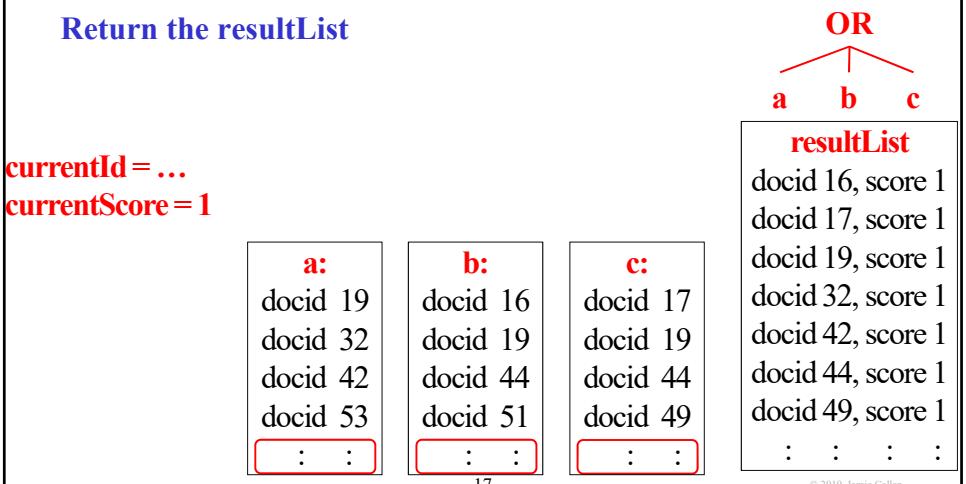
a:	b:	c:
docid 19	docid 16	docid 17
docid 32	docid 19	docid 19
docid 42	docid 44	docid 44
docid 53	docid 51	docid 49
: :	: :	: :

16

Document Retrieval: Document-at-a-Time (DAAT) Query Evaluation

Continue the loop until every inverted list is fully processed

Return the resultList



17

Document Retrieval: Document-at-a-Time (DAAT) Query Evaluation

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 decide which iterators to advance

A more efficient implementation combines loops

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

There are many opportunities for clever optimization

18

Document Retrieval: Document-at-a-Time (DAAT) Query Evaluation

How does DAAT support structured queries?

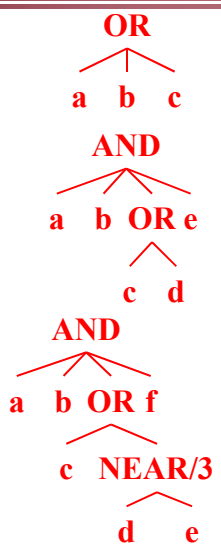
- Conceptually, it is something like this

```
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

The next lecture covers this in more detail



DAAT Query Evaluation 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 256MB 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

Document Retrieval: 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
- Document retrieval
  - TAAT
  - DAAT
- Query operators

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

- Later lectures explore these topics in greater detail

Query Operators

Usually search engines have rich query languages

- Query languages provide control over what is matched

Today's focus

- Types (classes) of query operators
  - There are many operators, but just a few types of operators
- The NEAR/n proximity operator

The goal is to prepare you for HW1

Three Types of Query Operators

1. Produce new inverted lists

- Dynamically create new index terms / concepts
- E.g., #SYNONYM, #NEAR

3. Combine scores

- Combine estimates about how well a document matches
- E.g., #AND, #OR, WSUM

Inverted list

df:	4356
docid:	42
tf:	2
locs:	14
	157
:	

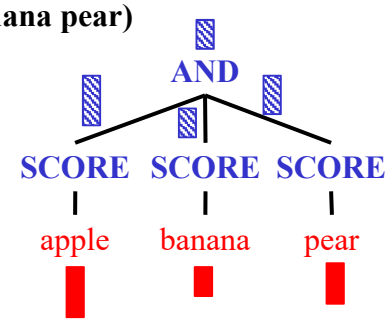
2. Use an inverted list to produce a score list

Score list

length:			94
docid:	14,	score:	3
docid:	89,	score:	2
docid:	127,	score:	4
:		:	

Three Types of Query Operators

#AND (apple banana pear)

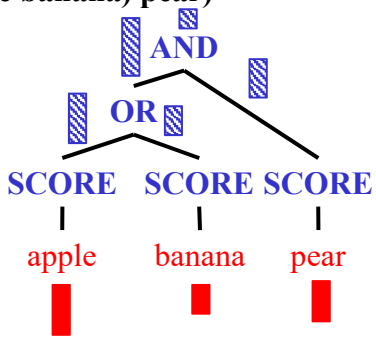


Inverted List Score List

25

Three Types of Query Operators

#AND (#OR (apple banana) pear)

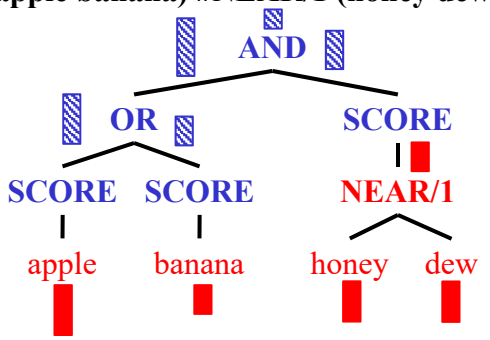


Inverted List Score List

26

Three Types of Query Operators

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



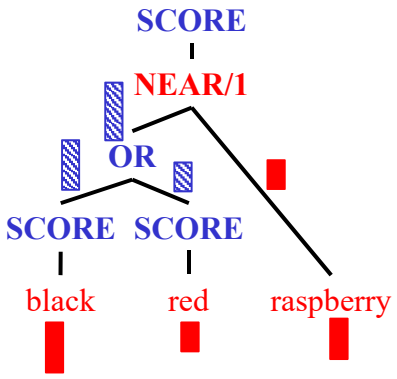
Inverted List **Score List**

27

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 (scores)

It does not produce locations

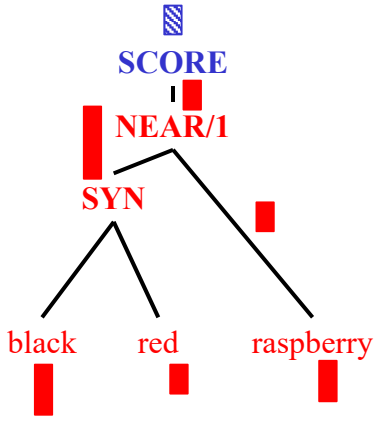
Inverted List **Score List**

28

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

The SYNONYM operator combines inverted lists

It creates an inverted list for a new concept

Inverted List **Score List**

29

OR vs SYN (SYNONYM)

OR and SYN are very different

SYN dynamically constructs new concepts (new inverted lists)

- By merging inverted lists
- The result is an inverted list

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

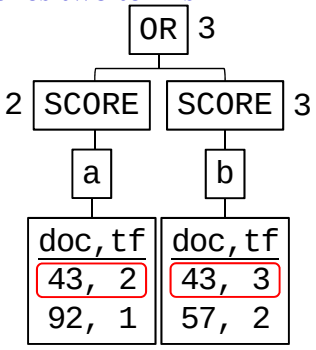
- Evidence obtained from matching multiple terms
- The result is a score list

These operators produce different search engine behavior

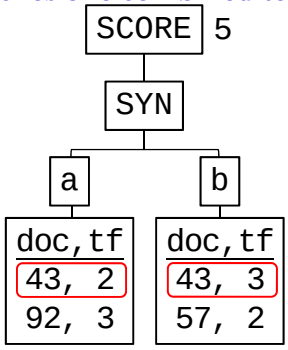
30

OR vs SYN (SYNONYM): Ranked Boolean Retrieval Model

#OR (a b)
Matches two terms



#SYN (a b)
Matches one combined term



SCORE: tf

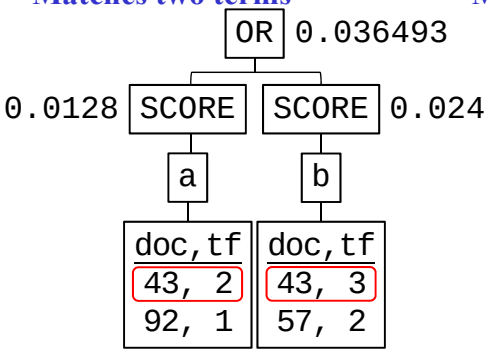
OR: MAX

#SYN (a b) > #OR (a b)

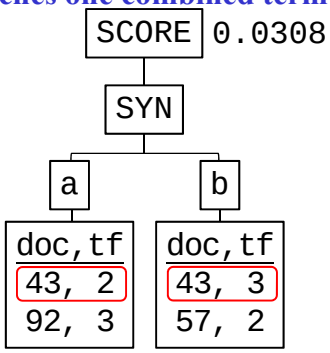
31

OR vs SYN (SYNONYM): Indri Retrieval Model

#OR (a b)
Matches two terms



#SYN (a b)
Matches one combined term



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

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

#OR (a b) > #SYN (a b)

32

Where Do SCORE Operators Come From?

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

The query parser inserts them automatically

- If the query operator expects a score list argument && its argument is an operator that produces inverted lists Then wrap the argument in a SCORE operator
- E.g., #AND (a b) → #AND (#SCORE (a) #SCORE (b))
- The QryEval homework software does this

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

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)

35

Proximity Operators: A Simple Greedy NEAR/n Algorithm

There are many ways to implement the NEAR/n operator

- An exact implementation has high computational complexity
- Most implementations are greedy and inexact

Typically proximity operators have complexity O(|C|)

- A single pass down each inverted list
- Similar in complexity to merging sorted lists
- They may not find some matches, but good enough for most tasks

Your implementation must match Jamie's greedy algorithm

36

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)

37

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

38

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)

Advance all doc iterators until they point to the same document

- **This is a simple nested loop**

39

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 iterators

40

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)

Advance loc iterators $q_{i>0}$ left-to-right such that loc (q_i) < loc (q_{i+1})

- **Not necessary here**

41

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)

Do a left-to-right check of locations

48 – 47 ≤ n (match)

Record match

- **Right-most matching loc (48)**

42

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)

Increment all loc iterators

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)

Advance loc iterators $q_{i>0}$ left-to-right such that loc (q_i) < loc (q_{i+1})

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)

Do a left-to-right check of locations

133 – 98 > 3 (no match)

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)

Increment q_0 loc iterator

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)

Advance loc iterators $q_{i>0}$ left-to-right such that loc (q_i) < loc (q_{i+1})

- **Not necessary here**

47

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)

Do a left-to-right check of locations

133 – 132 ≤ n (match)

Record match

- **Right-most matching loc (133)**

48

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)

Increment all loc iterators

q_0 locs are exhausted. No more matches are possible in this document

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)

Increment all doc iterators

…

Continue until the inverted lists are exhausted

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)

51

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

52

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)

Advance all doc iterators until they point to the same document

- **This is a simple nested loop**

53

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 iterators

54

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)

Advance loc iterators $q_{i>0}$ left-to-right such that loc (q_i) < loc (q_{i+1})

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)

Do a left-to-right check of locations

- **48 – 47 ≤ n (match)**
- **51 – 48 ≤ n (match)**

Record match

- **Right-most matching loc (51)**

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)

Increment all loc iterators

57

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)

Advance loc iterators $q_{i>0}$ left-to-right such that loc (q_i) < loc (q_{i+1})

58

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)

Do a left-to-right check of locations

133 – 98 > n (no match)

59

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)

Increment q_0 loc iterator

60

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)

Advance loc iterators $q_{i>0}$ left-to-right such that loc (q_i) < loc (q_{i+1}))

- **Not necessary here**

61

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)

Do a left-to-right check of locations

133 – 132 ≤ n (match)
137 – 133 > n (no match)

62

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)

Increment q_0 loc iterator

q_0 locs are exhausted. No more matches are possible in this document

63

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)

Increment all doc iterators

…

Continue until the inverted lists are exhausted

64

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 q_0's loc iterator to be advanced when this match failed

This is a flaw in the simple greedy algorithm.

Better algorithms are possible, but also more complex

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 {0, 1} and {13, 15}
- Results for the NEAR operator: tf =2, and locations=1, 15

Query: #NEAR/2 (a b c)

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

- There are two matches {0, 2, 4} and {1, 3, 5}
- Results for the NEAR operator: tf =2, and locations=4, 5

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/3 (a b c)

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

- The greedy algorithm fails to find a match
  - It considers {0, 1, 6} (the first location for each term)
  - {0, 1, 6} fails to match, so the q_0 location pointer advances
    - » a
  - The list for a is exhausted, so this text does not match
- A better algorithm would find a match at {0, 3, 6}
  - More accurate algorithms are much slower
  - The greedy algorithm is usually sufficient in practice
- Use the greedy algorithm for your homework

Proximity Operators: NEAR/n FAQ

Query: #NEAR/2 (a a b)

Text: **a b a a b**

- Our algorithm is not explicitly designed for duplicate arguments
- But … nothing prohibits duplicate arguments
- Probably it will work in most cases
  - E.g., it would match locations (0, 2, 4) above

These cases haven't been studied much by researchers

- But Google seems to support "apple apple pie"

Outline

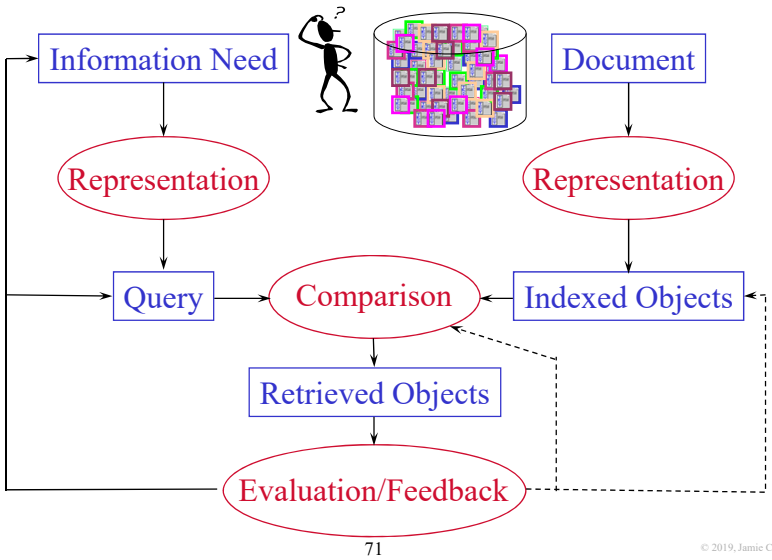
Introduction to…

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



71

Waitlist Reminder

If you are on the waitlist…

- I will admit people from the waitlist today or tomorrow
- Make sure that you have room in your schedule to be admitted

72