
**11-442 / 11-642 / 11-742:
Search Engines**

Document Structure

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

1

Outline

Document structure

- Index support for structure
- Fields
- Multiple representations of meaning
- Hierarchical structure (“XML documents”)

33

© 2021, Jamie Callan

33

Hierarchical Document Structure

- Hierarchical document structure raises some of the same issues that were seen with flat document structures**
 - Manual queries vs. automatic queries
 - Independent evidence vs. different perspectives
- Implemented using forward indexes or inverted lists that store document structure**
- Much of the work on hierarchical document structure has been done in Bayesian inference networks**
 - Indri

34

© 2021, Jamie Callan

35

Indexing Text Data under Space Constraints

Rishi Hiron
University of California Irvine
rhiron@uci.edu

Hakan Hacigumus
IBM Almaden Research
hahagan@us.ibm.com

Rishi Iyer
IBM Silicon Valley Lab
rishi@us.ibm.com

Shard Mehrotra
University of California Irvine
shardm@uci.edu

ABSTRACT

An important class of queries in the LIRE paradigm is RQL. In the absence of an index, LIRE queries are subject to performance degradation. The notion of indexing an substructure (or phrase) has been explored earlier without sufficient consideration of efficiency. A phrase is useful to phrase some queries that are not efficiently expressible in the query language. A phrase is a finite number of terms subject to storage constraints that gives restricted pointing for a given query workload. Our contributions include: (i) a formal problem definition, proof that the problem is NP-hard and adaptation of a previously studied approximate algorithm that produces results within a provable error bound; (ii) performance evaluation of the algorithm on the most realistic test data, and the parallelization of the algorithm, using compression and a parallel to handle larger queries.

Categories and Subject Descriptors: H.2.2 [Database]: Relational databases, Formal languages; H.2.3 [Database]: Indexing; Indexing methods; H.2.3 [Database]: Indexing; Indexing methods; H.2.3 [Database]: Indexing; Indexing methods

General Terms: Algorithms, Experimentation

Keywords: LIRE queries, RQL, Index, Index, Index

1. INTRODUCTION

In this paper we study the problem of designing efficient indexing techniques to support RQL LIRE queries over static data. In RQL, through the LIRE class, UNIX style, wild card queries can be specified. Two special characters, “*” and “%” are used to specify “any finite character string” and “any substructure” (including the empty string) respectively. LIRE queries are a subset of relational queries that have been extensively studied in the literature [19], [20], [21], [22], [23]. [24] provides an excellent survey of various data structures/algorithms developed for phrase queries. Unfortunately, none of these specific index data structures are supported as an access method by modern DBMS. Given the complexity of implementing some

data structures in DBMS (e.g., [25]), solutions employing existing access methods (already supported by DBMS - viz., B-trees [26], and B+ trees [27]) are more practical and better suited to the data oriented generalization of substructures. We explore a phrase based indexing approach in which queries are indexed based on a set of phrases they contain. A phrase is defined as follows:

Definition 1.1 (phrase): A phrase α is a string of length l over an arbitrary positive integer (usually small).

Note that in our case the phrase α is a LIRE query over a dataset (collection) of strings using a given based approach.

Evaluating LIRE queries using phrases: First, a set of phrases α are chosen to index a database of strings S . For each α , α is a phrase to verify string s if α contains a substructure α of s (denoted as $\alpha \subseteq s$). For all α in the database, α is a phrase if $\alpha \subseteq s$ for all s in the database, where α is the set of all substructures in S that are a phrase in S . It is easy to see that any string containing α also contains each of the phrases in α . Therefore, a important of strings containing α is a phrase in S . Finally, the last phrase is printed out to determine the result set.

Note that the phrase based technique for evaluating LIRE queries is straightforward, it takes several non trivial tasks that require deeper analysis:

- How should candidate phrases be generated and which of the candidate phrases should be chosen to build the index? It should be chosen the phrase of α that is the most specific and the least general which requires performance of queries.
- What data structure should be used to store the selected phrases and how should the phrases be processed given the data structure?

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made for distribution, for advertising or promotional purposes, for creating new collective works, or for resale.

Copyright 2004 ACM 978-1-55588-106-1/04/0001...\$5.00.

Hierarchical Document Structure

```

graph TD
    doc["document  
P(w|θdoc)"] --> title["title  
P(w|θtitle)"]
    doc --> body["body  
P(w|θbody)"]
    doc --> bib["bibliography  
P(w|θbib)"]
    body --> s1["section 1  
P(w|θsection 1)"]
    body --> s2["section 2  
P(w|θsection 2)"]
    body --> sn["section n  
P(w|θsection n)"]
    s1 --> st1["section title  
P(w|θsection title)"]
    s1 --> p1["paragraph 1  
P(w|θparagraph 1)"]
    s1 --> pn["paragraph n  
P(w|θparagraph n)"]
    
```

35

© 2021, Jamie Callan

35

Structured Documents: Hierarchical Structure

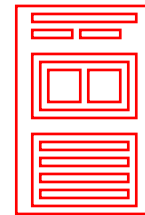
Documents with fields and multiple representations are simple uses of document structure

- Each element is independent of other elements
- A term occurs in the element that contains it
- **Retrieval goal:** Retrieve a document



Documents with hierarchical fields are a more advanced use of document structure

- Elements are related to other elements
- A term occurs in all elements that contain it
- **Retrieval goal:** Retrieve documents or elements

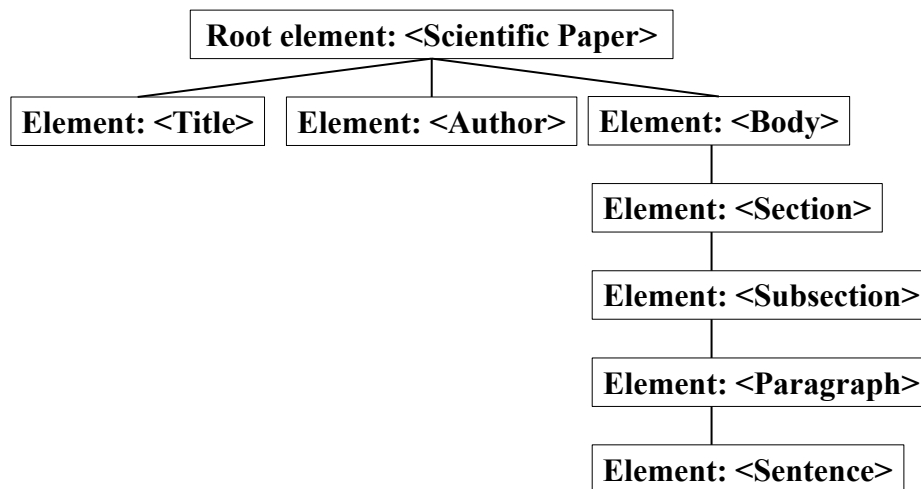


36

© 2021, Jamie Callan

36

Structured Documents: Example Hierarchical Structures

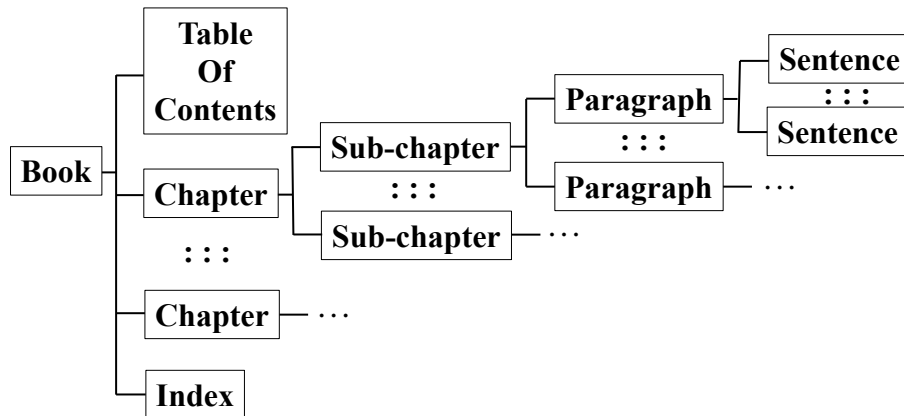


37

© 2021, Jamie Callan

37

Structured Documents: Example Hierarchical Structures

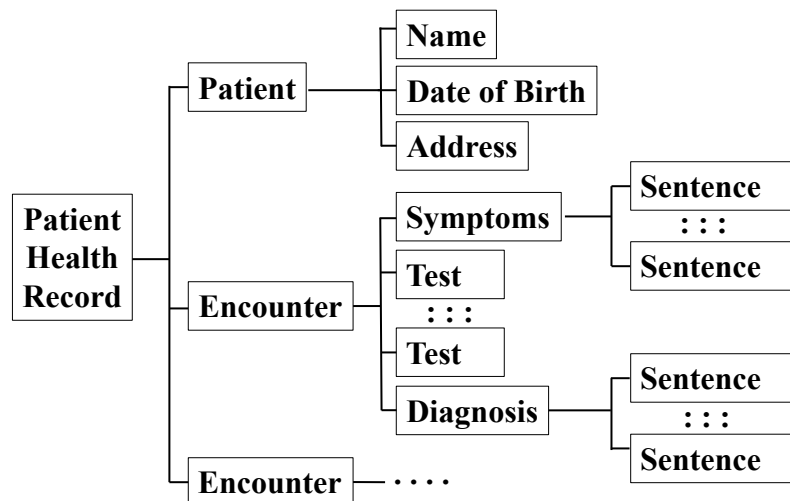


38

© 2021, Jamie Callan

38

Structured Documents: Example Hierarchical Structures



39

© 2021, Jamie Callan

39

Structured Documents: New Issues Raised by Hierarchical Structure

What type of element to retrieve?

- Document? Encounter? Diagnosis?

What corpus statistics to use?

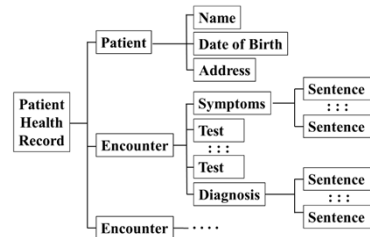
- Document vs. element

How to combine evidence from different elements?

- If the query is “chest pain”, should we give higher rank to patients that have several matching sentences or encounters?

Exact-match vs. best-match document structure?

- Perhaps the query doesn’t exactly match the document structure



40

© 2021, Jamie Callan

40

Hierarchical Structure: Ranking Elements



Indri allows the ranked element to be specified in the query

- Any belief (“score”) operator can rank any type of element
- Syntax: #OPERATOR[element](argument ...)
- If no element is specified, the default is Document

Queries that rank different types of elements

- Documents: #AND(breast cancer treatment)
- Paragraphs: #WAND[paragraph](3 breast 4 cancer 2 treatment)
- Sentences: #AND[sentence](breast cancer treatment)

The result is a list of elements and scores

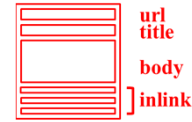
- I.e., a score list (element_i, score_i)

41

© 2021, Jamie Callan

41

Hierarchical Structure: Ranking Elements



How is element e ranked?

- I.e., what is $p(q_i | e)$?

One option: Use Jelinek-Mercer smoothing

$$p(q_i | e) = (1 - \lambda)p_{MLE}(q_i | e) + \lambda p_{MLE}(q_i | C)$$

Typically Dirichlet smoothing isn't used for element ranking

- Probably more a historical accident than a justified choice

42

© 2021, Jamie Callan

42

Hierarchical Structure: Ranking Elements



Common problems in ranking elements

- The document structure is wrong (or broken)
 - Common with web documents
- The query is too strict
 - `#AND[title](iphone)` doesn't match "Apple Cuts Phone Price"
 - `#AND[paragraph](solutions to poverty)`
 - "poverty" and "solutions" may appear in different paragraphs

Additional smoothing reduces the effect of these problems

$$p(q_i | e) = \lambda_1 p_{MLE}(q_i | e) + \lambda_2 p_{MLE}(q_i | d) + \lambda_3 p_{MLE}(q_i | C) \quad \lambda_1 + \lambda_2 + \lambda_3 = 1$$

Element Document Collection

43

© 2021, Jamie Callan

43

Hierarchical Structure: Documents with Multiple Elements



Often the goal is to use evidence from one type of element (e_1) to rank another type of element (e_2)

- E.g., Find web pages that have ‘Jamie Callan’ as inlink text

How is evidence from different types of elements combined?

1. Aggregation
 - Can be done during indexing (e.g., Lucene)
 - Can be done during query evaluation (e.g., Indri)
2. Combination
 - Can be done during query evaluation (e.g., Indri)

44

© 2021, Jamie Callan

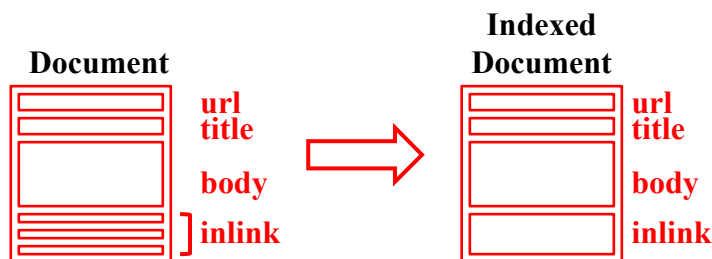
44

Hierarchical Structure: Aggregation During Indexing



Goal: Find web pages that have ‘apple ipad’ as inlink text

- Use information from inlinks to produce a document score



- Combine (“aggregate”) multiple instances of an element type (e.g., inlink) into one bag of words
- Okapi BM25F and Lucene do this

45

© 2021, Jamie Callan

45

Hierarchical Structure: Aggregation During Querying



Goal: Find web pages that have ‘apple ipad’ as inlink text

- Use information from inlinks to produce a document score

Indri queries can aggregate element information into document information during query evaluation

- apple.inlink aggregates all occurrences of apple in inlink elements
 - Implemented as a QryIop operator
 - Result is one inverted list for apple as an inlink term

Example

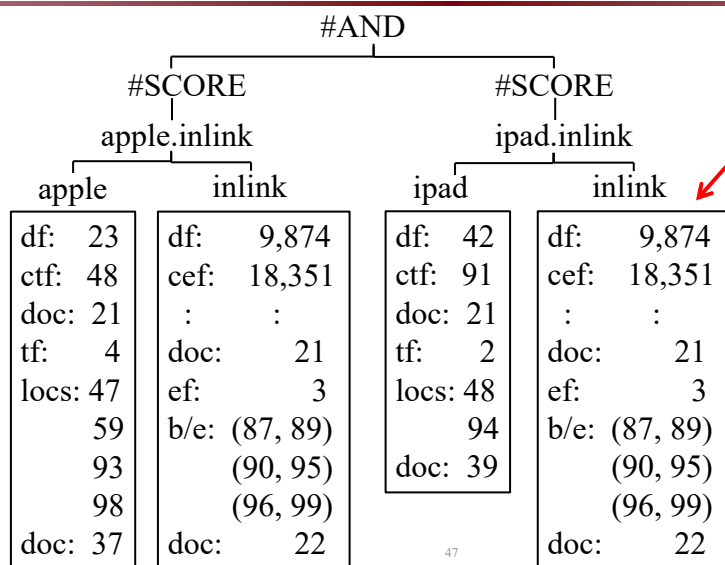
```
#WAND( #WSUM (0.3 apple.title 0.2 apple.inlink 0.5 apple.body )
        #WSUM (0.3 ipad.title 0.2 ipad.inlink 0.5 ipad.body ) )
```

46

© 2021, Jamie Callan

46

Aggregated Elements in Indri: #AND(apple.inlink ipad.inlink)



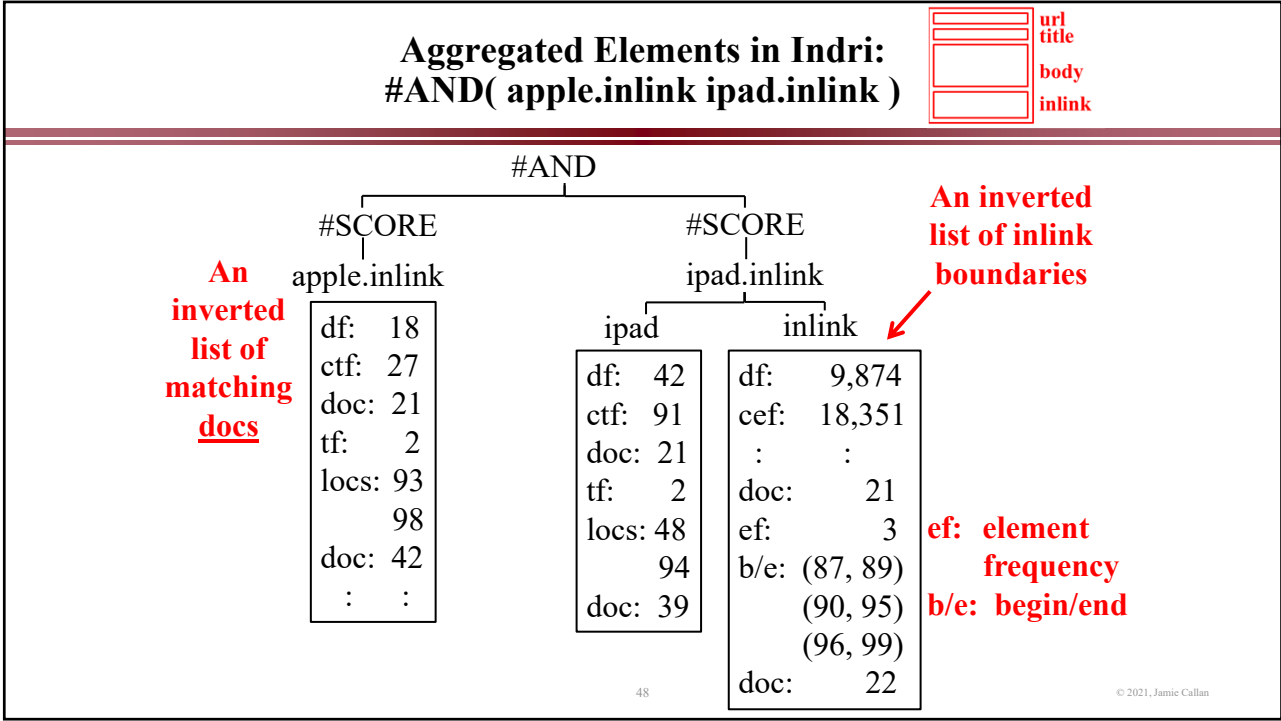
**An inverted
list of inlink
boundaries**

**ef: element
frequency
b/e: begin/end**

47

© 2021, Jamie Callan

47



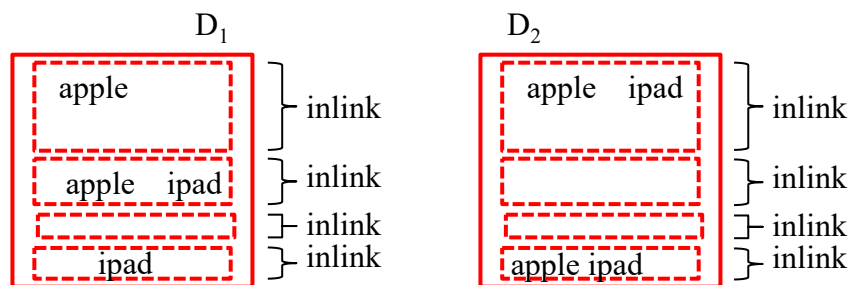
Hierarchical Structure: Aggregated Elements



Goal: Rank documents that have inlinks containing apple ipad

#AND(apple.inlink ipad.inlink)

- Aggregation makes these documents look the same to the query
 - Probably okay, because inlinks are short and typically used to rank documents



50

© 2021, Jamie Callan

50

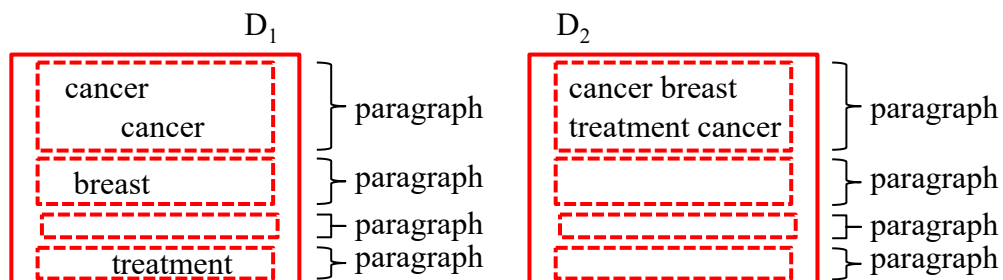
Hierarchical Structure: Aggregated Elements



Goal: Rank documents that have paragraphs that discuss breast cancer treatment

#AND(breast.paragraph cancer.paragraph treatment.paragraph)

- Aggregation makes these documents look the same to the query
 - Probably not okay, e.g., if returning paragraphs to the user



51

© 2021, Jamie Callan

51

Hierarchical Structure: Combining Evidence



The alternative to aggregation is combining evidence from different elements of the same type

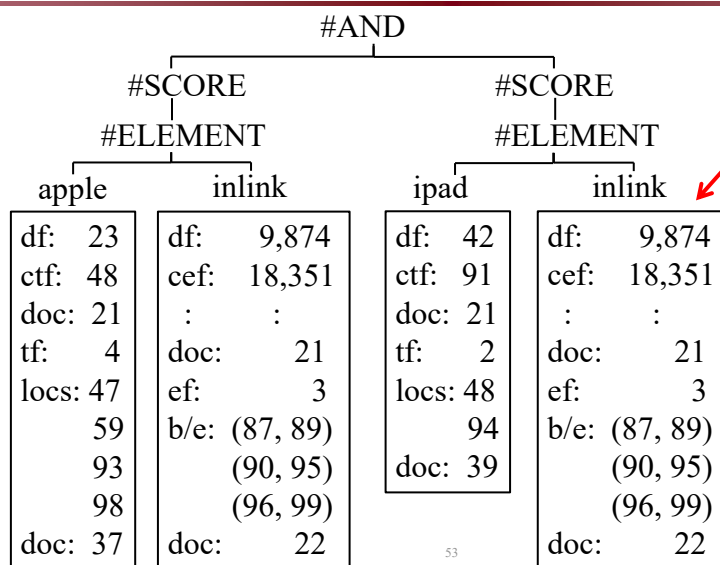
- **Aggregation:** Combine inverted lists
 - Use QryIop operators
- **Combination:** Combine scores
 - Use both QryIop and QrySop operators

52

© 2021, Jamie Callan

52

Combining Elements in Indri: #AND[inlink](apple ipad)



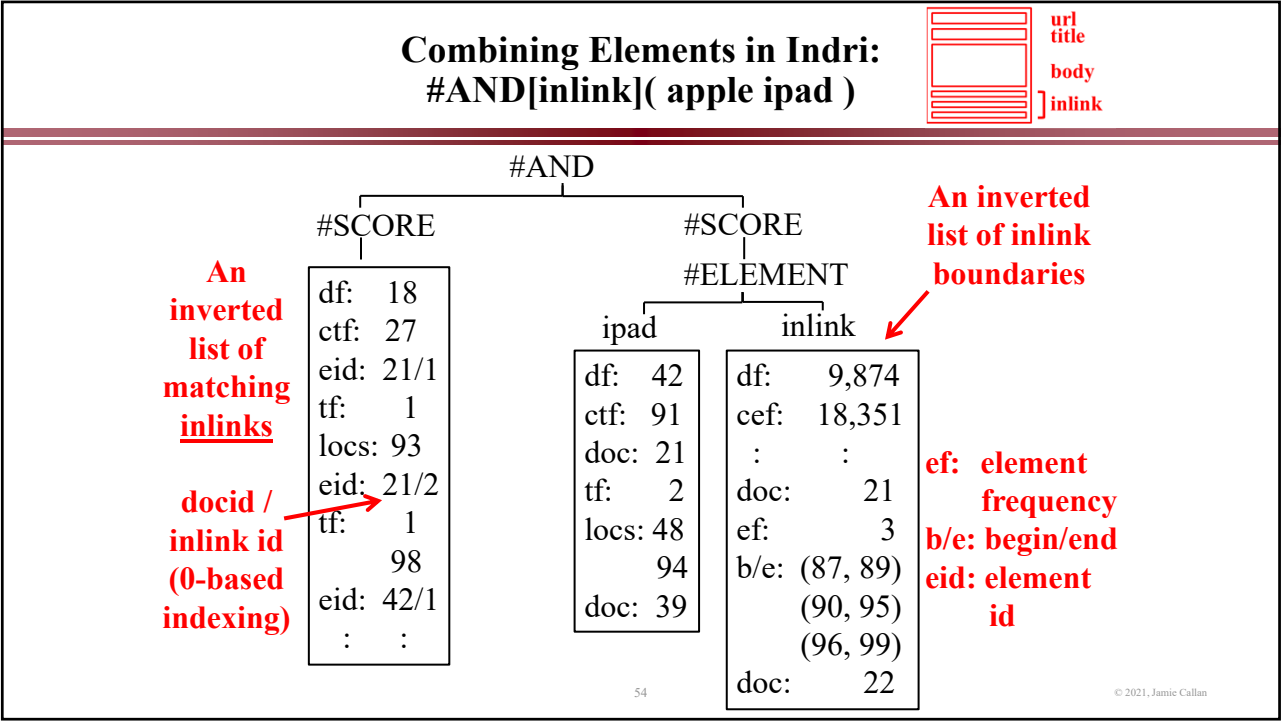
An inverted list of inlink boundaries

ef: element frequency
b/e: begin/end

53

© 2021, Jamie Callan

53



Combining Elements in Indri:
#AND[inlink](apple ipad)

url
title
body
inlink

#AND

A
score
list of
matching
inlinks

df: 18
ctf: 27
21/1, 0.453
21/2, 0.467
42/1, 0.421
: :

docid /
inlink id

#SCORE

df: 25
ctf: 41
eid: 21/1
tf: 1
locs: 94
eid: 57/3
: :

eid: element
id

56

© 2021, Jamie Callan

56

Combining Elements in Indri:
#AND[inlink](apple ipad)

url
title
body
inlink

#AND

A
score
list of
matching
inlinks

df: 18
ctf: 27
21/1, 0.453
21/2, 0.467
42/1, 0.421
: :

docid /
inlink id

df: 25
ctf: 41
21/1, 0.673
57/3, 0.597
: :

57

© 2021, Jamie Callan

57

Page 13

13

Combining Elements in Indri: #AND[inlink](apple ipad)



**A
score
list of
matching
inlinks**

df:	25
ctf:	41
21/1,	0.305
21/2,	0.006
42/1,	0.005
57/3,	0.014
:	:

**docid / score
inlink id**

If the goal is to retrieve a ranked list of inlinks, sort the list by score and display it

- Done!

58

© 2021, Jamie Callan

58

Combining Elements in Indri

The previous example is a little unusual

- Retrieving individual inlinks is an unusual task
- Purpose: To provide a clear contrast with aggregation

The technique in the previous example is general and useful

- E.g., retrieving sentences or passages for Alexa
- E.g., retrieving sections of a long scientific paper

59

© 2021, Jamie Callan

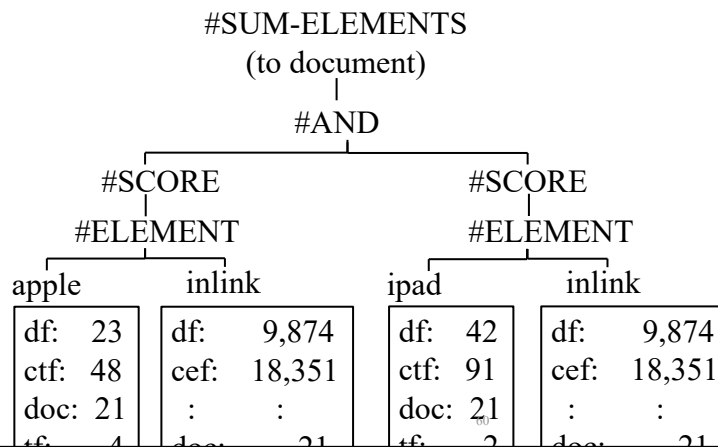
59

Hierarchical Structure: Combining Element Scores



Suppose the goal is to use inlink scores to rank documents

#SUM[document] (#AND[inlink](apple ipad))



© 2021, Jamie Callan

60

Hierarchical Structure: Combining Element Scores



Suppose the goal is to use inlink scores to rank documents

#SUM[document] (#AND[inlink](apple ipad))

#SUM-ELEMENTS
(to document)

df:	25
ctf:	41
21/1,	0.305
21/2,	0.006
42/1,	0.005
57/3,	0.014
:	:

**The result of
#AND[inlink](apple ipad)
(shown on earlier slides)**

61

© 2021, Jamie Callan

61

Hierarchical Structure: Combining Element Scores



Suppose the goal is to use inlink scores to rank documents

#SUM[document] (#AND[inlink](apple ipad))

#SUM-ELEMENTS
(to document)

df:	25
ctf:	41
21/1,	0.305
21/2,	0.006
42/1,	0.005
57/3,	0.014
:	:

The result of
#AND[inlink](apple ipad)
(shown on earlier slides)

#SUM-ELEMENTS
sums child scores (inlink)
to produce parent scores (document)

62

© 2021, Jamie Callan

62

Hierarchical Structure: Combining Element Scores



Suppose the goal is to use inlink scores to rank documents

#SUM[document] (#AND[inlink](apple ipad))

#SUM-ELEMENTS
(to document)

df:	25
ctf:	41
21/1,	0.305
21/2,	0.006
42/1,	0.005
57/3,	0.014
:	:

Two inlinks
for docid 21

The result of
#AND[inlink](apple ipad)
(shown on earlier slides)

#SUM-ELEMENTS
sums child scores (inlink)
to produce parent scores (document)

63

© 2021, Jamie Callan

63

Hierarchical Structure: Combining Element Scores



Suppose the goal is to use inlink scores to rank documents

#SUM[document] (#AND[inlink](apple ipad))

Docid 21 {

df:	25
ctf:	32
	21, 0.311
	42, 0.005
	57, 0.014
:	:

64

© 2021, Jamie Callan

64

Hierarchical Structure: Combining Element Scores



Indri queries must specify the combination method

- The right method is problem-specific

Return documents that have many matching sentences

- #MAX[document] (#AND[sentence](breast cancer treatment))
 - Only the best sentence is considered
- #SUM[document] (#AND[sentence](breast cancer treatment))
 - Poorly matching sentences reduce the score
- #OR[document] (#AND[sentence](breast cancer treatment))
 - Prefer documents with more matching sentences

65

© 2021, Jamie Callan

65

Hierarchical Structure: Combining Element Scores



What does this query mean?

#SUM[document] (#AND[sentence] (breast cancer treatment))

Your software uses depth-first evaluation

1. Apply #AND (breast cancer treatment) to every sentence
 - The result is a list of <sentence, score>
2. Apply #SUM to the sentence scores produced in step 1.
 - Note: Different #SUM operator semantics than HW2 #SUM
 - The score for a document is the sum of its sentence scores
 - The result is <document, score>

The query returns a list of <document, score>

66

© 2021, Jamie Callan

66

Hierarchical Structure: Combining Element Scores



Consider 4 chapters that have sentences matching the query ...which is the best match?

- #MAX considers them equal
- #AND prefers C₄
- #OR prefers C₁
- #AVERAGE prefers C₄

C ₁	C ₂	C ₃	C ₄
0.010	0.010	0.010	0.010
0.009	0.009	0.009	
0.001	0.001		
0.001			

It may seem “obvious” that #OR is the best choice
...but, the best choice is problem-dependent

67

© 2021, Jamie Callan

67

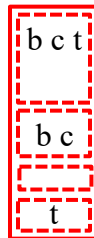


Hierarchical Structure: Elements in Indri

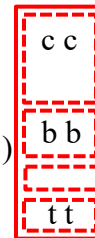


Suppose the goal is to retrieve a document that has paragraphs discussing breast cancer treatment

#OR[document] (
 #AND[paragraph] (
 breast
 cancer
 treatment))



#AND[document] (
 breast.paragraph
 cancer.paragraph
 treatment.paragraph)



Combination: Partial credit for paragraphs that partially match

Aggregation: The terms might not be in the same paragraph

68

© 2021, Jamie Callan

68

Outline

Document structure

- Index support for structure
- Fields
- Multiple representations of meaning
- Hierarchical structure (“XML documents”)

69

© 2021, Jamie Callan

69

Summary

Fields

- Used for two very different purposes
 - Independent evidence (author, title, journal, ...)
 - Multiple (related) representations (url, title, body, ...)
- Know the difference
- Know how these are supported by each retrieval model
- Know how to these would be used differently in queries

70

© 2021, Jamie Callan

70

Summary

Hierarchical structure

- How to combine evidence when several elements match
- Exact-match vs. best-match document structure
- Know how these are supported by Indri
- Know how to these would be used differently in queries

71

© 2021, Jamie Callan

71

For Additional Information

- S. Robertson and H. Zaragoza. The Probabilistic Relevance Framework: BM25 and Beyond. *Foundations and Trends in Information Retrieval*, 3(4). 2009.
- G. Salton. *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer*. Addison-Wesley. 1989.
- S. Walker, S.E. Robertson, M. Boughanem, G.J.F. Jones, K. Sparck Jones. “Okapi at TREC-6: Automatic ad hoc, VLC, routing, filtering, and QSDR. TREC-6 Proceedings. 1997.