11-442 / 11-642:
Search Engines

Overview of the QryEval Software

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

Outline

- **QryEval overview**
- **Query evaluation**
  - The Qry class
  - Iteration
  - Matching
  - Calculating scores
- **Overview of query parsing**

QryEval

QryEval is a software application that conducts experiments

- Read a parameter file
  - Example parameter file for HW1

```
indexPath=someDirectory/index-gov2
retrievalAlgorithm=UnrankedBoolean
queryFilePath=queries.txt
trecEvalOutputPath=HW1-queries-UB.teIn
trecEvalOutputLength=100
```

  - Each homework will have additional parameters

QryEval

QryEval is a software application that conducts experiments

- Read a parameter file
- Read a query file (one query per line)

```
10:#OR(cheap internet)
26:#AND(lower heart rate)
71:living in india
```

QryEval

QryEval is a software application that conducts experiments

- Read a parameter file
- Read a query file (one query per line)
  - Parse the query

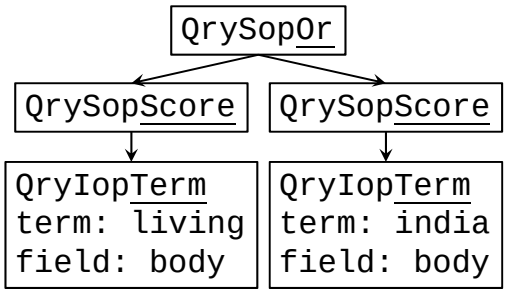
```
Qry q = QryParser.getQuery (
            "#or(living in india)" );
```

  - More on query parsing later in the lecture …

QryEval

QryEval is a software application that conducts experiments

- Read a parameter file
- Read a query file (one query per line)
  - Parse the query ("living in india")



Qry class
An abstract class
for all query operators

QrySopXxx subclasses
Sop: Score operator
Create a score list

QryIopXxx subclasses
Iop: Inverted list operator
Get an inverted list

QryEval

QryEval is a software application that conducts experiments

- Read a parameter file
- Read a query file (one query per line)
  - Parse the query
  - Evaluate the query, using a DAAT architecture

```
q.initialize (model);    // E.g., UnrankedBoolean

while (q.docIteratorHasMatch (model)) {
  int docid = q.docIteratorGetMatch ();
  double score = ((QrySop) q).getScore (model);
  result.add (docid, score);
  q.docIteratorAdvancePast (docid);
}
```

7

QryEval

QryEval is a software application that conducts experiments

- Read a parameter file
- Read a query file (one query per line)
  - Parse the query
  - Evaluate the query
  - Write the results for the query to a file

```
11 Q0 GX270-76-5299838 1   3.000  HW1a
11 Q0 GX000-25-2008761 2   2.000  HW1a
11 Q0 GX000-72-8784276 3   2.000  HW1a
```

Query Id | **Always Q0** | **Doc External Id** | **Doc Rank** | **Doc Score** | **Run ID (your choice)**

8

QryEval

QryEval is a software application that conducts experiments

Main software classes

- RetrievalModel: Define the model and parameters (if any)
- QryParser: Parse a text query into a query tree
- Qry: Create and evaluate query operators
  - QryIopXxx Inverted list operators (e.g., QryIopNear)
  - QrySopXxx Score list operators (e.g., QrySopAnd)
- Idx: Access the index
- InvList: Create and access inverted lists
- ScoreList: Create and access score lists
- TermVector: Forward index (we haven't covered this yet)

9

Outline

- QryEval overview
- **Query evaluation**
  - The Qry class
  - Iteration
  - Matching
  - Calculating scores
- Overview of query parsing

10

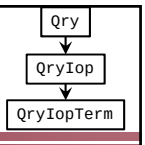
Qry

Qry is the base class for all query operators

- Data that <u>every</u> query operator has (e.g., query arguments)
- Methods that work for <u>all</u> query operators (e.g., appendArg)
- Methods that <u>every</u> query operator must define (e.g., HasMatch)
- It has two subclasses: QryIop and QrySop
  - QrySop: The base class for operators that return scores
    - » SCORE, AND, OR, SUM, WAND, WSUM, …
  - QryIop: The base class for operators that return inverted lists
    - » TERM, NEAR, WINDOW

11

Parts of an Object May Be Defined By Different Parts of the Class Hierarchy



QryIopTerm object: An object for storing query terms

```
String displayName;
ArrayList<Qry> args;
boolean matchStored;
int matchingDocid;
-----------------------
InvList invertedList;
int docIteratorIndex;
Int locIteratorIndex;
String field;
-----------------------
String term;
```

Defined by Qry

Defined by QryIop

Defined by QryIopTerm

<u>Defined</u> in multiple places, but <u>stored</u> in one place … the object

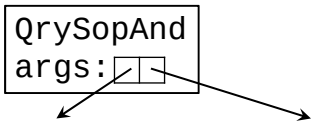
- Each object has its own args, invertedList, term string, …

12

Qry

Qry is the base class for all query operators

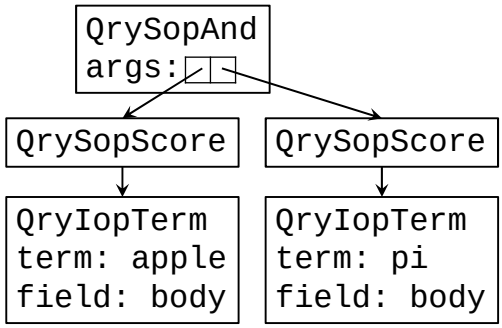
- It defines a place to store query operator arguments & other data
  - Conceptually: #AND (apple pi)
  - Actually:



Qry

Qry is the base class for all query operators

- It controls how arguments are appended to the query operator
  - E.g., automatically insert a #SCORE operator between a scoring (Sop) and an inverted list (Iop) operator



Qry

Qry is the base class for all query operators

- It defines docIterators to iterate over matching documents

```
docIteratorHasMatch     // Each subclass defines
      :     :     :
docIteratorGetMatch     // Get from HasMatch cache
docIteratorAdvancePast  // Advance past a docid
docIteratorAdvanceTo    // Advance to a docid
```

  – These are <u>not</u> Java-style iterators

15

Qry

Qry is the base class for all query operators

- It defines docIterators to iterate over matching documents

```
docIteratorHasMatch     // Each subclass defines
```

  – There are a few standard matching styles that are implemented as utility methods

```
docIteratorHasMatchAll // Matches all args
docIteratorHasMatchMin // Matches min docid
```

  – When you implement `docIteratorHasMatch` for a new [query operator, retrieval model] pair, consider whether one of the standard utility methods meets your needs

    » E.g., `docIteratorHasMatchMin` for an OR operator

16

Iteration

Query evaluation is divided into three parts

1. Get all inverted lists
2. Get the docid of the next document that matches the query
3. Get the score of docid (which must be a matching document)

Iteration

Query evaluation is divided into three parts

1. Get all inverted lists
   - Done during <u>query initialization</u>
   - There are two ways of obtaining inverted lists
     - » Read from disk, e.g., "apple"
     - » Construct dynamically, e.g., "#NEAR/3 (lady gaga)"
   - Assumption: Everything fits into RAM
     - » This is a simple system for homework
     - » A production system might process inverted lists in blocks to control memory usage

Iteration

Query evaluation is divided into three parts

1. Get all inverted lists
2. Get the docid of the next document that matches the query
   - Iterate over (actual) inverted lists and (virtual) score lists
   - There are only a few matching strategies
     - » any query argument matches the document ("union")
     - » all query arguments match the document ("intersection")
   - The retrieval model determines what is considered a match
     - » E.g., Ranked Boolean: AND must match <u>all</u> arguments
     - » E.g., Indri: AND must match <u>at least one</u> argument

19

Iteration

Query evaluation is divided into three parts

1. Get all inverted lists
2. Get the docid of the next document that matches the query
3. Get the score of docid (which must be a matching document)
   - The retrieval model determines how the score is calculated

20

Iteration

The retrieval model determines how a query operator iterates

- Ranked Boolean AND does it differently from an Indri AND

However, there are a few "typical" styles of iteration

- E.g., HasMatchFirst, HasMatchAll, HasMatchMin
- These are defined in Qry.java
- Often individual query operators just call one of the standard methods
  - Before you implement something, consider whether a standard method will meet your needs

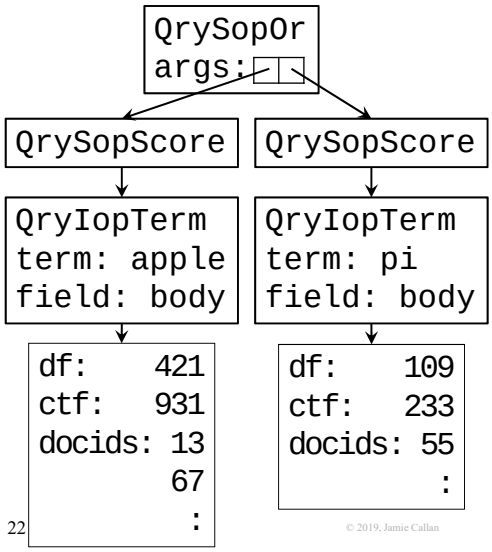
Method QrySopOr.docIteratorHasMatch

The OR operator matches if <u>any</u> argument matches

It uses docIteratorHasMatchMin

- Iterate over the arguments
- Ask each argument to return its current docid
- Return the minimum docid

Recursion handles complex queries naturally



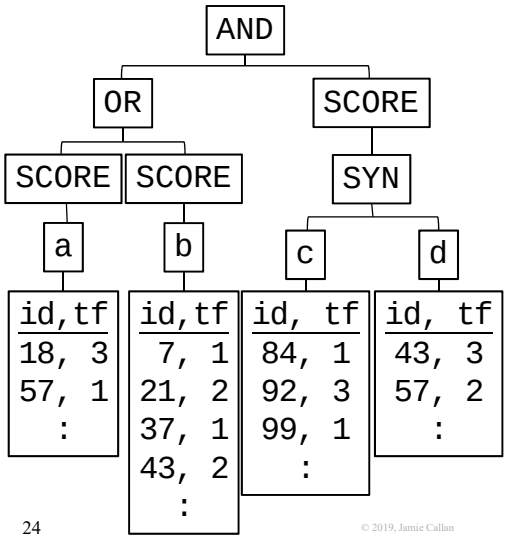
Caching

When a docIteratorHasMatch matches a document, it caches the docid to improve efficiency

- docIteratorGetMatch reads the docid from the cache
- Why not have HasMatch just return the docid?
  - If there is no match, it would need to return an invalid docid or throw an exception
  - Those seem messier to me

Matching and Scoring

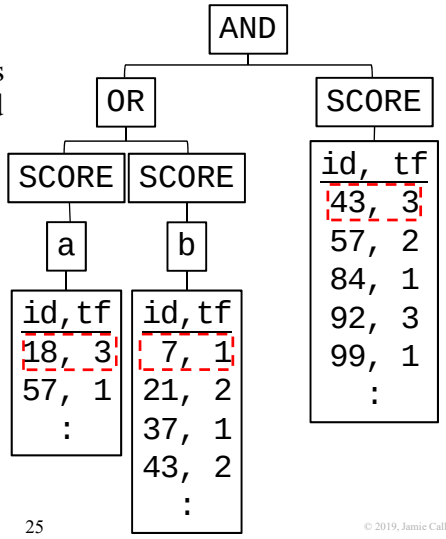
What is the first document that matches this query?



Matching and Scoring

Call q.initialize()

- Inverted list query operators (e.g., SYN) are materialized
  - Converted to inverted lists
- Iterators are initialized

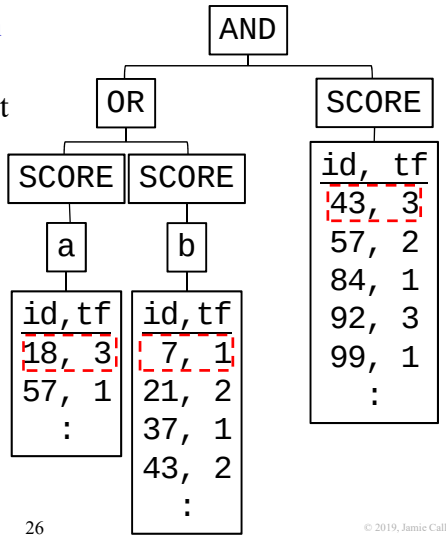


25

Matching and Scoring

Call q.docIteratorHasMatch on the AND operator

- The query is structured, so it passes the request to its children
- The children are query operators, so they pass it to their children
- Eventually the requests reach inverted lists

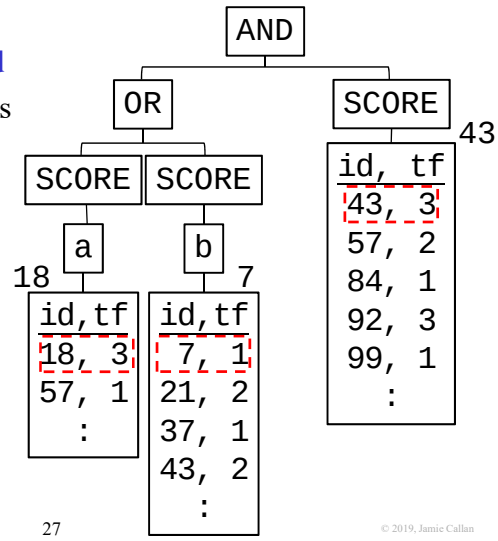


26

Matching and Scoring

The inverted lists return their next matching docid

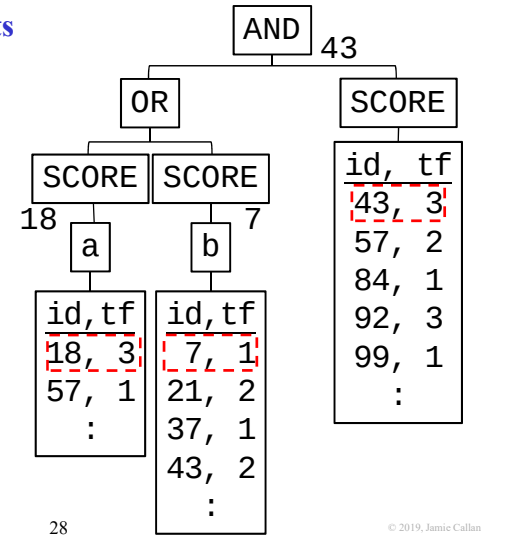
- The ids get passed upwards



27

Matching and Scoring

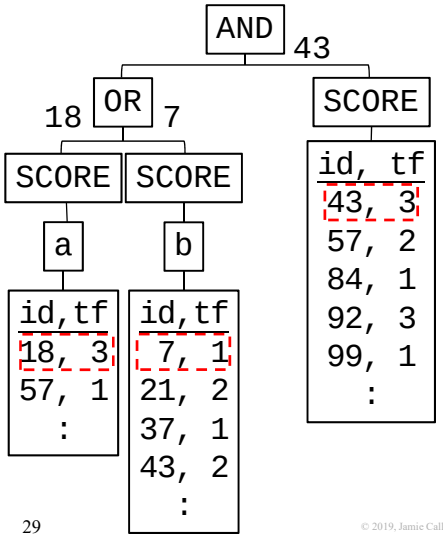
Each operator calculates its next matching document



28

Matching and Scoring

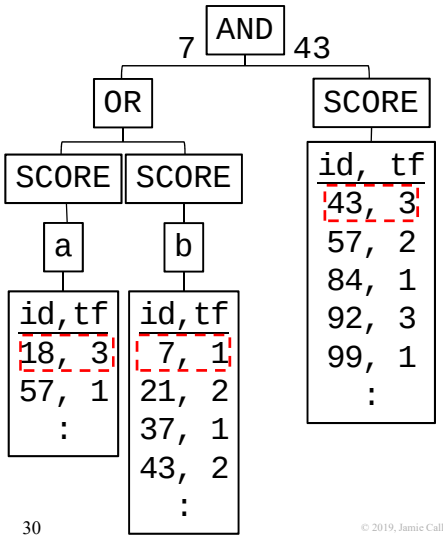
Each operator calculates its next matching document



29

Matching and Scoring

Each operator calculates its next matching document

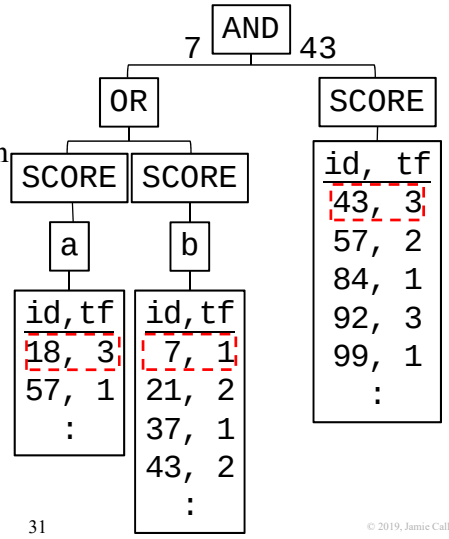


30

Matching and Scoring

Each operator calculates its next matching document

- AND doesn't have a match
- It knows that the next match must have docid ≥ 43
- It tells all children to advance their iterators to docid 43 (or the next docid after 43)

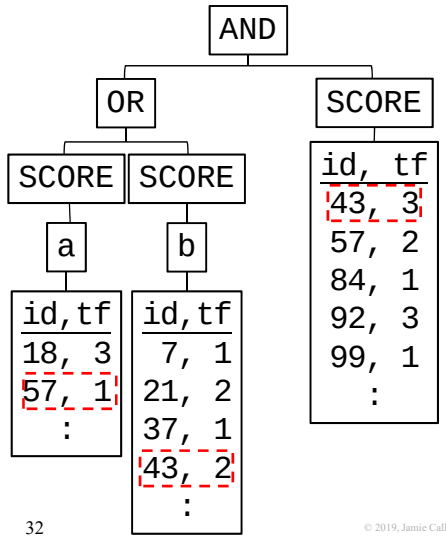


31

Matching and Scoring

AND polls its children again (this is a while loop)

- The children are query operators, so they pass it to their children
- Eventually the requests reach inverted lists

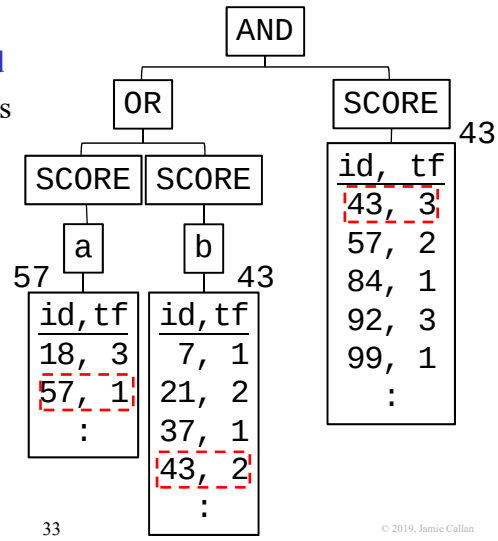


32

Matching and Scoring

The inverted lists return their next matching docid

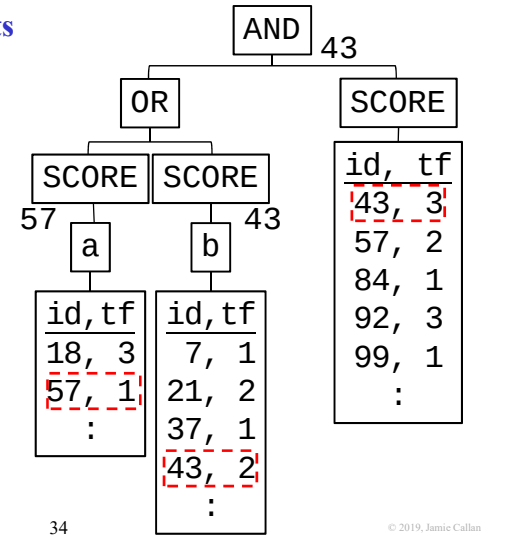
- The ids get passed upwards



33

Matching and Scoring

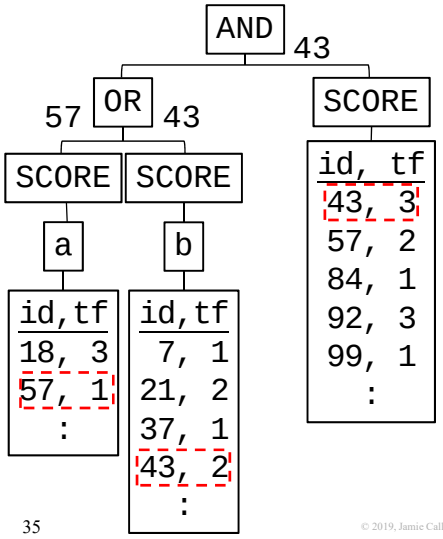
Each operator calculates its next matching document



34

Matching and Scoring

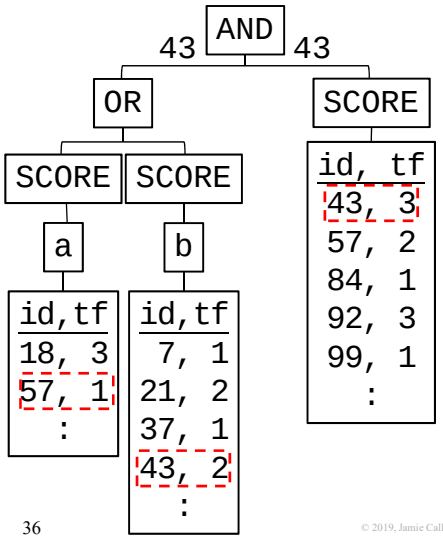
Each operator calculates its next matching document



35

Matching and Scoring

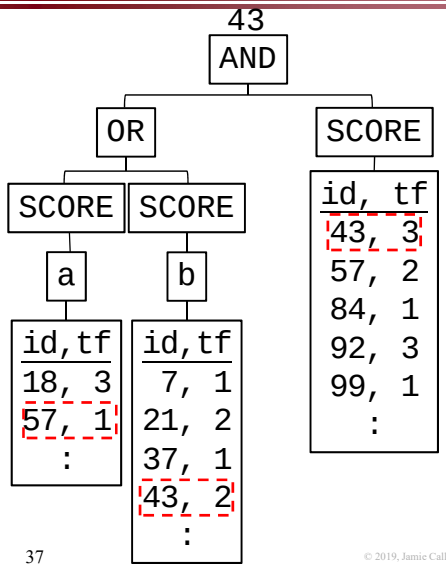
Each operator calculates its next matching document



36

Matching and Scoring

Each operator calculates its next matching document

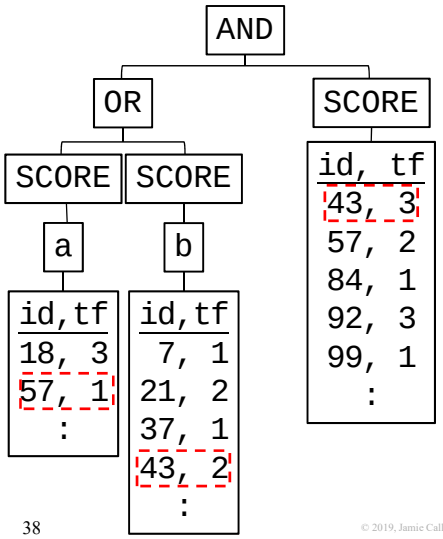


37

Matching and Scoring

Call q.getScore on the AND operator

- AND has cached the fact that docid 43 matches
- The query is structured, so AND passes the request to its children
- The children are query operators, so they pass it to their children
- Eventually the requests reach SCORE operators

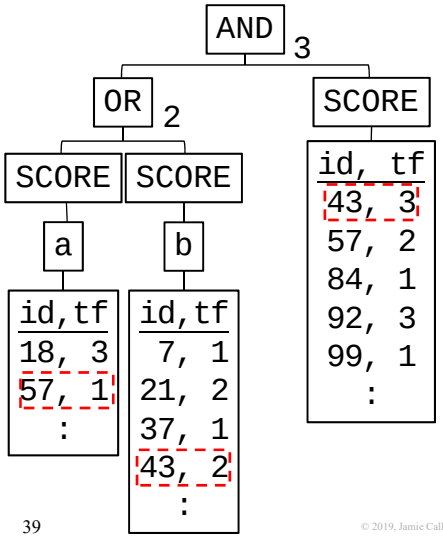


38

Matching and Scoring

The SCORE operators return scores

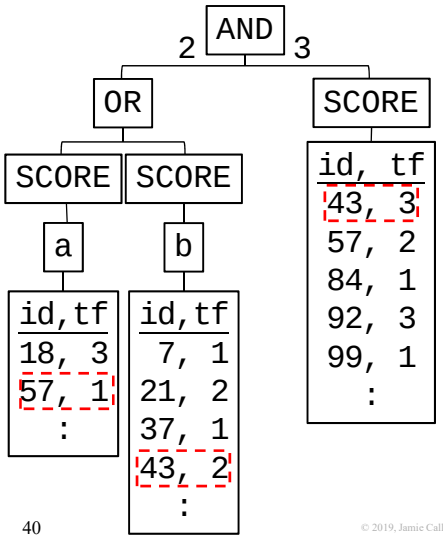
- Assume ranked Boolean



39

Matching and Scoring

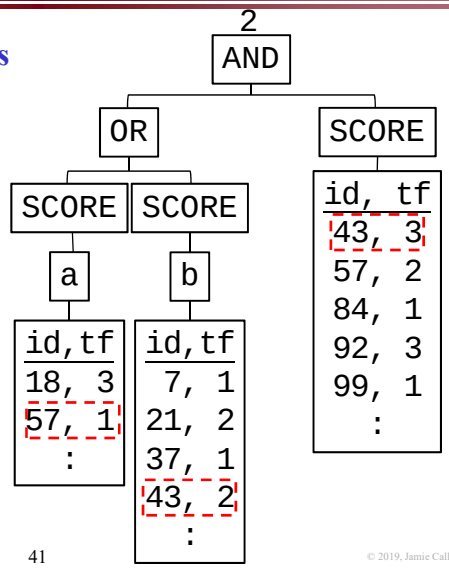
The OR operator combines scores



40

Matching and Scoring

The AND operator combines scores

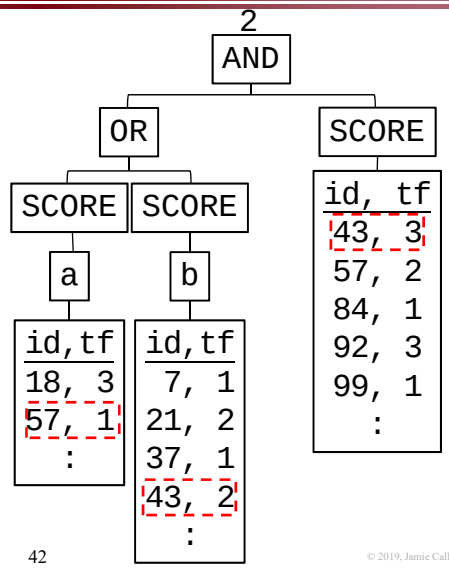


41

Matching and Scoring

Easy!

Repeat for the next match



42

Calculating Scores

Each score operator (QrySopXxx) implements getScore (RetrievalModel r)

- Traverse the query to calculate a score for the current docid
  - We know that it matches, so just calculate a score
- The retrieval model tells the operator what strategy to use for calculating the score
  - For HW1, RankedBoolean and UnrankedBoolean
    - » Unranked Boolean: Score is 1.0 for all matches
    - » Ranked Boolean: Score is > 0.0 for all matches
  - For HW2, BM25 and Indri
  - Retrieval models for HW2 will also store parameters

43

Qry Class Summary

The Qry class implements DAAT scoring

- Iterate over (actual) inverted lists and (virtual) score lists
- Several general ways to match a query operator to a document
  - Match all arguments, any argument, …
- Allows you to add different ways to calculate document scores
  - Unranked boolean, ranked boolean, …
- Much use of inheritance and recursion
  - Minimizes redundant implementation ☺
  - Requires greater understanding ☹

44

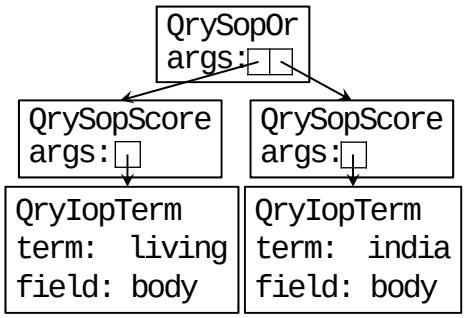
Outline

- QryEval overview
- Query evaluation
  - The Qry class
  - Iteration
  - Matching
  - Calculating scores
- **Overview of query parsing**

Query Parsing

QryParser is a simple query parser

```
Qry q = QryParser.getQuery (
            "#or(living in india)" );
```



Query Parsing

Pop the operator from the string

- Leftmost token, starts with '#'
- Create the operator (e.g., QrySopOr)

```
#OR(a b c)

#AND(a #OR(b c) d)
```

Find the list of query arguments

- Delimited by leftmost '(' and its matching ')'
- For each argument
  - – If it is not a stopword
    - » If it is a term, create a term object (i.e., QryIopTerm)
    - » Otherwise, recursively call the query parser
    - » Add the result (a Qry object) to the operator argument list
      - • A #SCORE operator may be inserted automatically

47

Query Parsing

```
#AND(a b c)
```

48

Query Parsing

Allocate an AND operator.

```
#AND(a b c)
```

operator

```
QrySopAnd
args:
```


Query Parsing

```
#AND(a b c)
```

Left '(' and right ')'

```
QrySopAnd
args:
```

Query Parsing

#AND(a b c)

arguments

```
QrySopAnd
args:
```


Query Parsing

#AND(a b c)

arguments

Allocate a TERM operator. Append it as an argument to the AND operator.

```
QrySopAnd
args:
```

```
QrySopScore
args:
```

```
QryIopTerm
term:  a
field: body
```

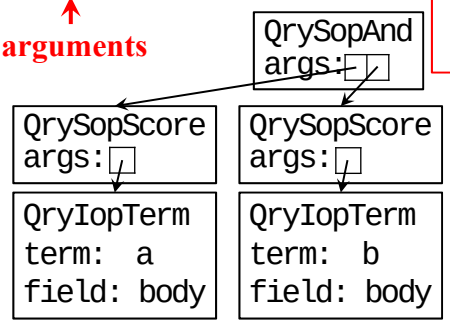
Note: Qry.appendArg automatically inserts #SCORE operators

Query Parsing

#AND(a b c)

arguments

Allocate a TERM operator. Append it as an argument to the AND operator.



Note: Qry.appendArg automatically inserts #SCORE operators

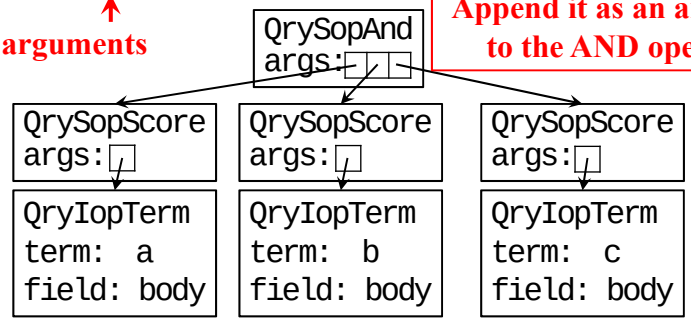
53

Query Parsing

#AND(a b c)

arguments

Allocate a TERM operator. Append it as an argument to the AND operator.



Note: Qry.appendArg automatically inserts #SCORE operators

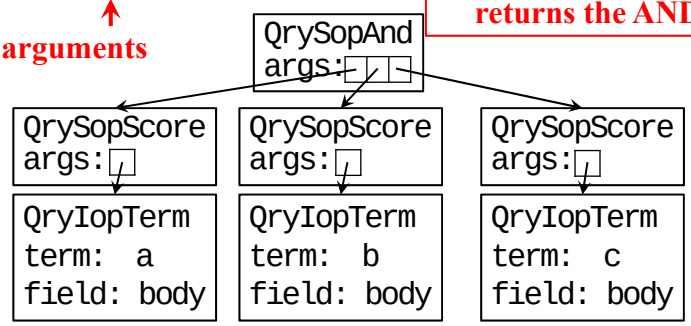
54

Query Parsing

#AND(a b c)

arguments

Query parsing completes & returns the AND node.



55

Query Parsing

#AND(a #OR(b c) d)

56

Query Parsing

`#AND(a #OR(b c) d)`

operator

Allocate an AND operator.

QrySopAnd
args:

Query Parsing

`#AND(a #OR(b c) d)`

Left '(' and right ')'

QrySopAnd
args:

Query Parsing

```
#AND(a #OR(b c) d)
```

arguments

```
QrySopAnd
args:
```


Query Parsing

```
#AND(a #OR(b c) d)
```

arguments

```
QrySopAnd
args:
```

Allocate a TERM operator. Append it as an argument to the AND operator.

```
QrySopScore
args:
```

```
QryIopTerm
term:  a
field: body
```

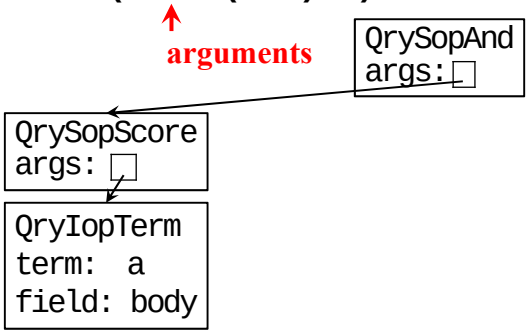
Note: Qry.appendArg automatically inserts #SCORE operators

Query Parsing

#AND(a #OR(b c) d)

arguments

Operator detected. Recursive call on the argument.



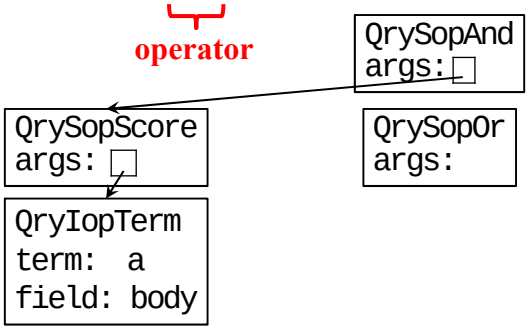
61

Query Parsing

#AND(a #OR(b c) d)

operator

Recursive call. Allocate an OR operator.



62

Query Parsing

Recursive call.

```
#AND(a #OR(b c) d)
```

Left '(' and right ')'

```
QrySopAnd
args:
```

```
QrySopScore
args:
```

```
QrySopOr
args:
```

```
QryIopTerm
term:  a
field: body
```

63

Query Parsing

Recursive call.

```
#AND(a #OR(b c) d)
```

arguments

```
QrySopAnd
args:
```

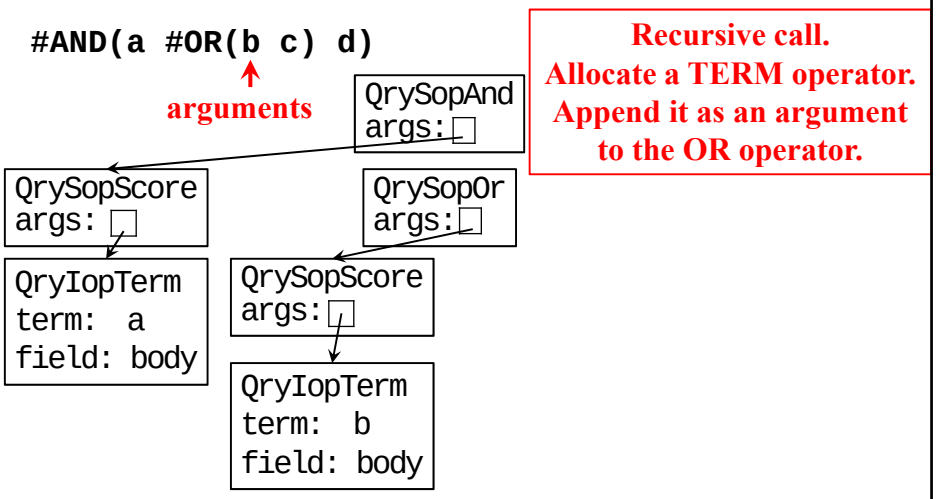
```
QrySopScore
args:
```

```
QrySopOr
args:
```

```
QryIopTerm
term:  a
field: body
```

64

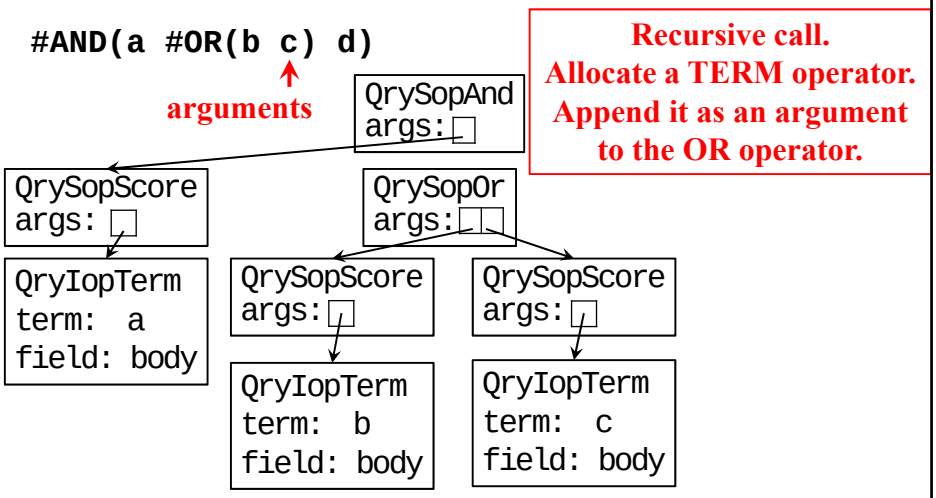
Query Parsing



Note: Qry.appendArg automatically inserts #SCORE operators

65

Query Parsing



Note: Qry.appendArg automatically inserts #SCORE operators

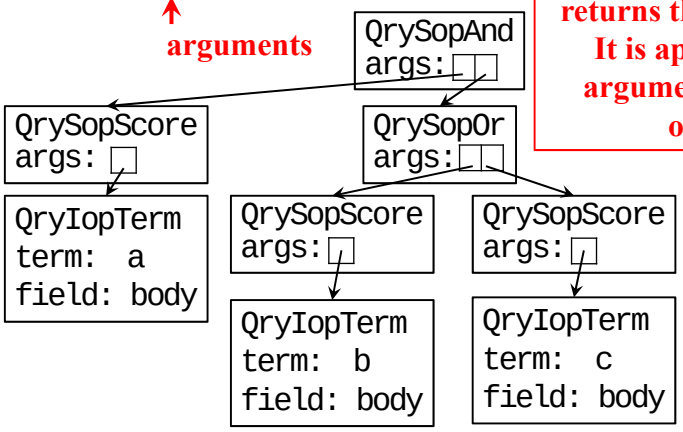
66

Query Parsing

#AND(a #OR(b c) d)

arguments

Recursive call completes & returns the OR operator. It is appended as an argument to the AND operator.

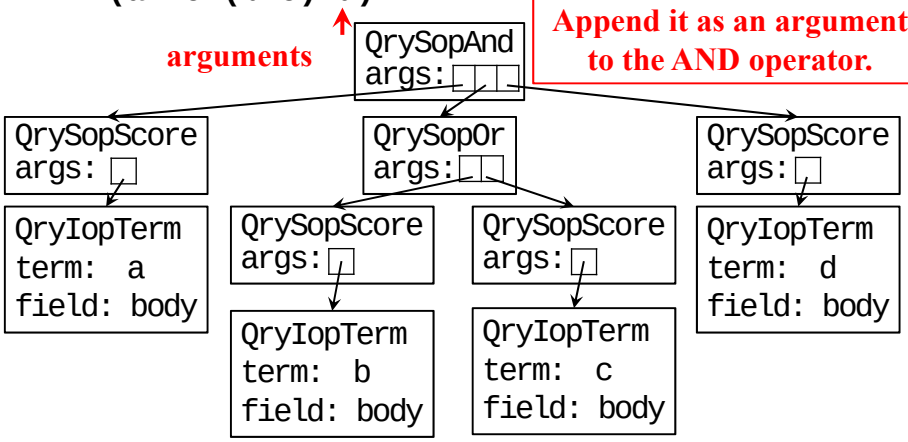


Query Parsing

#AND(a #OR(b c) d)

arguments

Allocate a TERM operator. Append it as an argument to the AND operator.

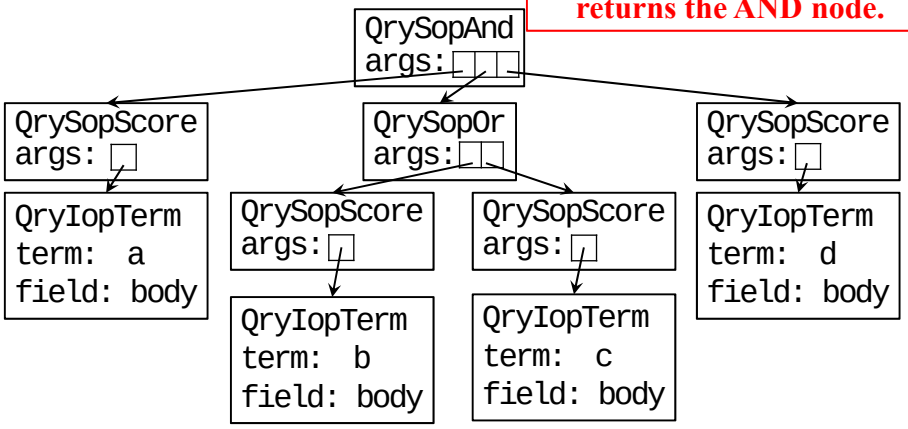


Note: Qry.appendArg automatically inserts #SCORE operators

Query Parsing

#AND(a #OR(b c) d)

Query parsing completes & returns the AND node.



69

Query Parsing

a b c

70

Query Parsing

`a b c`

This is a syntax error because there is no query operator

We want the search engine to support unstructured queries

- **Solution:** The caller must add the default query operator
- E.g., **`a b c`** → **`#OR(a b c)`**

Every retrieval model has a default query operator

- But it isn't the same for every retrieval model
- The query parser doesn't know which retrieval model will be used
- So, the query parser can't apply the default query operator for you
- Your code must do it

71

Query Parsing

You will need to modify the query parser

- HW1: Add new query operators (e.g., #AND, #NEAR/n)
- HW2: Add support for query operators that require weights
  - E.g., #WSUM (0.5 barack 1.0 obama)

72

Outline

- **QryEval overview**
- **Query evaluation**
  - The Qry class
  - Iteration
  - Matching
  - Calculating scores
- **Overview of query parsing**