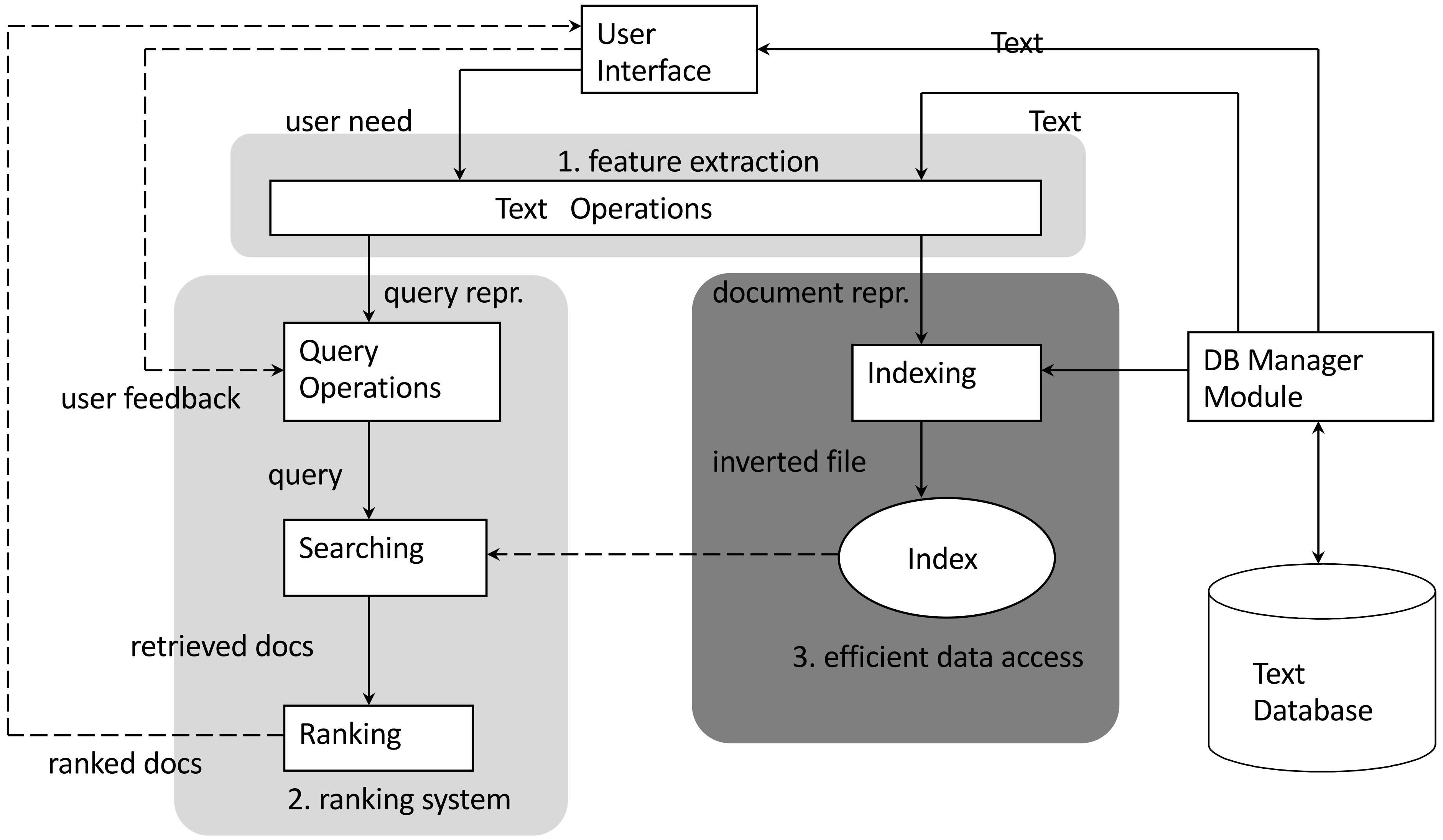
7. INDEXING FOR INFORMATION RETRIEVAL

Architecture of Text Retrieval Systems



Term Search

Problem: text retrieval algorithms need to find words in documents efficiently

- Boolean, probabilistic and vector space retrieval
- Given index term k_i, find document d_j

application $\longrightarrow$

B3, B17 $\longleftarrow$

B1 A Course on Integral Equations
B2 Attractors for Semigroups and Evolution Equations
B3 Automatic Differentiation of Algorithms: Theory, Implementation, and Application
B4 Geometrical Aspects of Partial Differential Equations
B5 Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra
B6 Introduction to Hamiltonian Dynamical Systems and the N-Body Problem
B7 Knapsack Problems: Algorithms and Computer Implementations
B8 Methods of Solving Singular Systems of Ordinary Differential Equations
B9 Nonlinear Systems
B10 Ordinary Differential Equations
B11 Oscillation Theory for Neutral Differential Equations with Delay
B12 Oscillation Theory of Delay Differential Equations
B13 Pseudodifferential Operators and Nonlinear Partial Differential Equations
B14 Sinc Methods for Quadrature and Differential Equations
B15 Stability of Stochastic Differential Equations with Respect to Semi-Martingales
B16 The Boundary Integral Approach to Static and Dynamic Contact Problems
B17 The Double Mellin-Barnes Type Integrals and Their Applications to Convolution Theory

Inverted Files

An inverted file is a word-oriented mechanism for indexing a text collection in order to speed up the term search task

- Addressing of documents and word positions within documents
- Most frequently used indexing technique for large text databases
- Appropriate when text collection is large and semi-static

Inverted Files

Inverted list l_k for a term k

$$l_k = \left[f_k \colon d_{i_1}, \dots, d_{i_{f_k}}\right]$$

- f_k number of documents in which k occurs
- $d_{i1},\dots,d_{ifk}$ list of document identifiers of documents containing k

Inverted File: lexicographically ordered sequence of inverted lists

$$IF = \left[i, k_i, l_{k_i}\right], i = 1, \dots, m$$

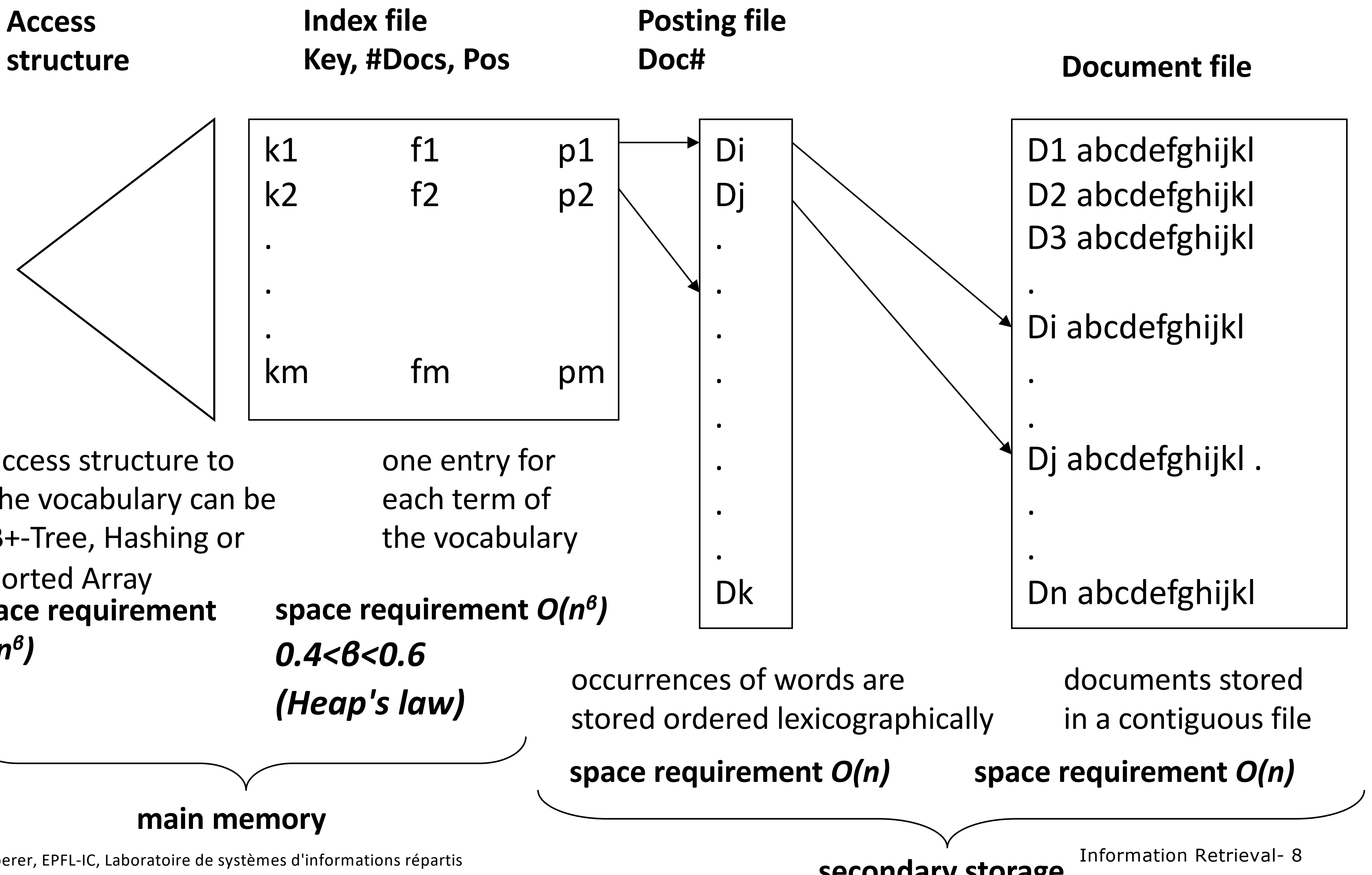
Example: Documents

B1 A Course on Integral Equations
B2 Attractors for Semigroups and Evolution Equations
B3 Automatic Differentiation of Algorithms: Theory, Implementation, and Application
B4 Geometrical Aspects of Partial Differential Equations
B5 Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra
B6 Introduction to Hamiltonian Dynamical Systems and the N-Body Problem
B7 Knapsack Problems: Algorithms and Computer Implementations
B8 Methods of Solving Singular Systems of Ordinary Differential Equations
B9 Nonlinear Systems
B10 Ordinary Differential Equations
B11 Oscillation Theory for Neutral Differential Equations with Delay
B12 Oscillation Theory of Delay Differential Equations
B13 Pseudodifferential Operators and Nonlinear Partial Differential Equations
B14 Sinc Methods for Quadrature and Differential Equations
B15 Stability of Stochastic Differential Equations with Respect to Semi-Martingales
B16 The Boundary Integral Approach to Static and Dynamic Contact Problems
B17 The Double Mellin-Barnes Type Integrals and Their Applications to Convolution Theory

Example

```
1   Algorithms      3  :  3   5   7
2   Application     2  :  3   17
3   Delay           2  :  11  12
4   Differential    8  :  4   8   10  11  12  13  14  15
5   Equations       10 :  1   2   4   8   10  11  12  13  14  15
6   Implementation  2  :  3   7
7   Integral        2  :  16  17
8   Introduction    2  :  5   6
9   Methods         2  :  8   14
10  Nonlinear       2  :  9   13
11  Ordinary        2  :  8   10
12  Oscillation     2  :  11  12
13  Partial         2  :  4   13
14  Problem         2  :  6   7
15  Systems         3  :  6   8   9
16  Theory          4  :  3   11  12  17
```

Physical Organization of Inverted Files



access structure to the vocabulary can be B+-Tree, Hashing or Sorted Array

space requirement *O($n^β$)*

one entry for each term of the vocabulary

space requirement *O($n^β$)*

$0.4<β<0.6$

(Heap's law)

occurrences of words are stored ordered lexicographically

space requirement *O(n)*

documents stored in a contiguous file

space requirement *O(n)*

Heap’s Law

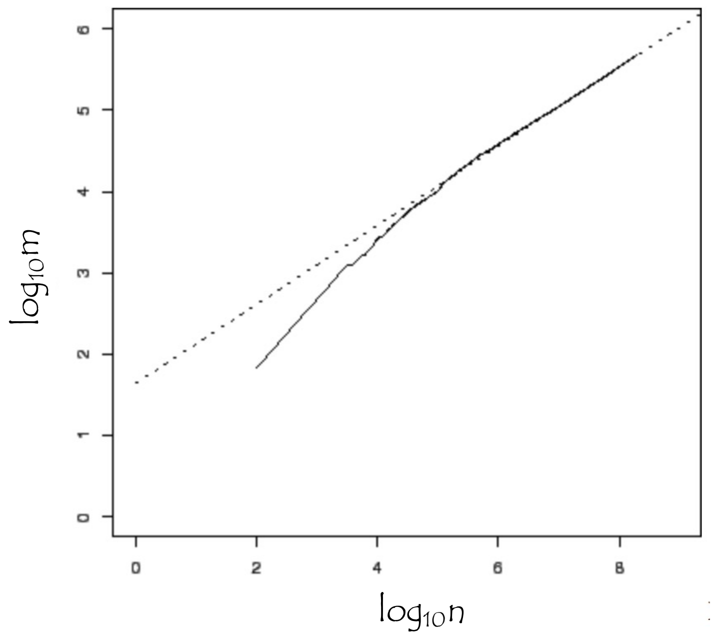
An empirical law that describes the relation between the size of a collection and the size of its vocabulary

$$m = kn^{\beta}$$

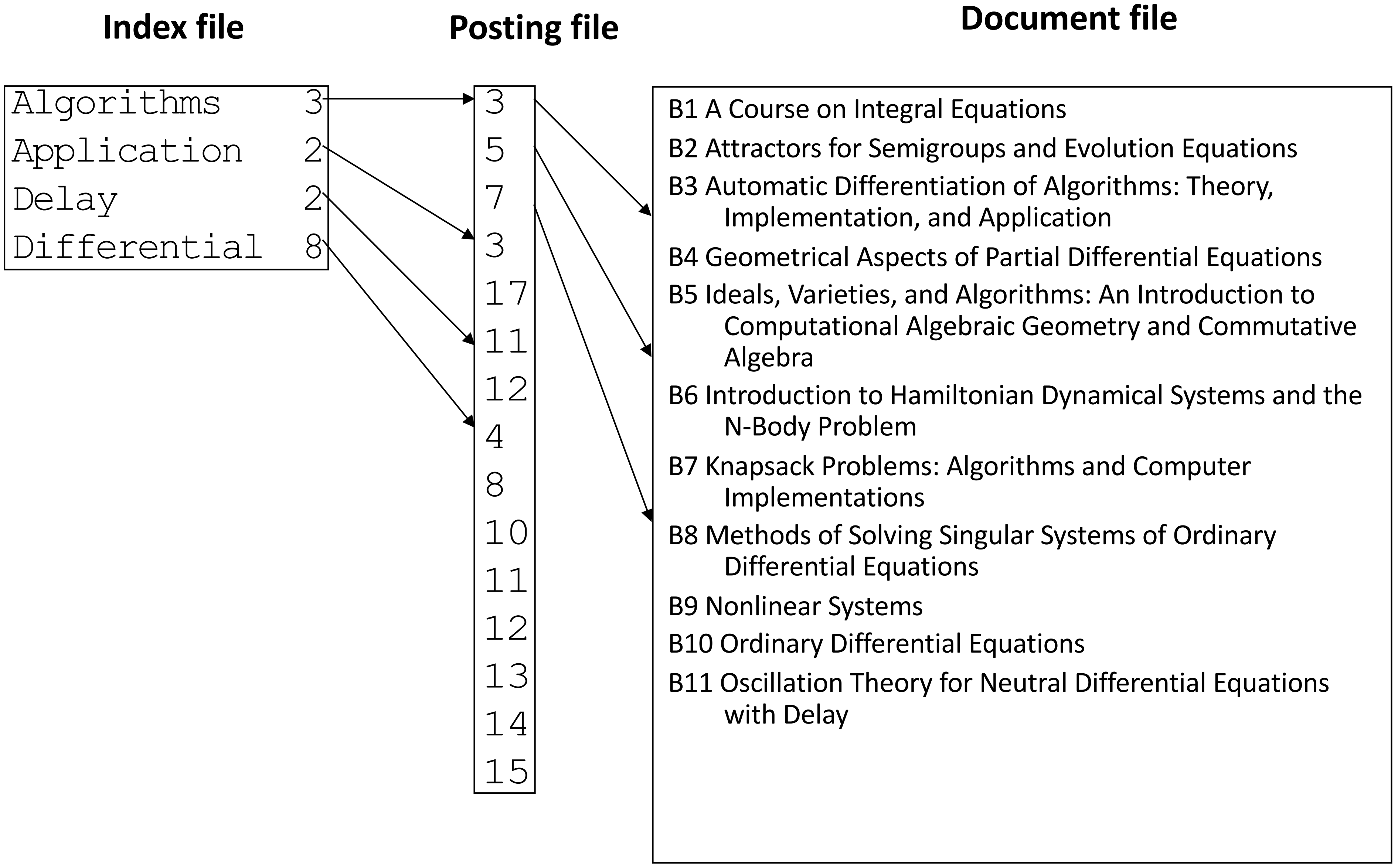
Typical values observed: $\beta \approx 0.5, 30 < k < 100$

Parameters depend on collection type and preprocessing

- Stemming, lower case decrease vocabulary size
- Numbers, spelling errors increase



Example



B1 A Course on Integral Equations
B2 Attractors for Semigroups and Evolution Equations
B3 Automatic Differentiation of Algorithms: Theory, Implementation, and Application
B4 Geometrical Aspects of Partial Differential Equations
B5 Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra
B6 Introduction to Hamiltonian Dynamical Systems and the N-Body Problem
B7 Knapsack Problems: Algorithms and Computer Implementations
B8 Methods of Solving Singular Systems of Ordinary Differential Equations
B9 Nonlinear Systems
B10 Ordinary Differential Equations
B11 Oscillation Theory for Neutral Differential Equations with Delay

Addressing Granularity

By default the entries in postings file point to (are) the document identifier

Other granularities can be used

- Pointing to a specific position with in the document
  - Example: (B1, 4) points to “Integral”
- Grouping multiple documents to one
  - Example: create groups of 4 documents, then G1 points to the group (B1,B2,B3,B4)

A posting indicates…

1. The frequency of a term in the vocabulary
2. The frequency of a term in a document
3. The occurrence of a term in a document
4. The list of terms occurring in a document

When indexing a document collection using an inverted file, the main space requirement is implied by ...

1. The access structure
2. The vocabulary
3. The index file
4. The postings file

Searching the Inverted File

Step 1: Vocabulary search

- the words present in the query are searched in the *index file*

Step 2: Retrieval of occurrences

- the lists of the occurrences of all words found are retrieved from the *posting file*

Step 3: Manipulation of occurrences

- the occurrences are processed in the *document file* to process the query

Construction of the Inverted File – Step 1

Step 1: Search phase

- The vocabulary is kept in an ordered data structure, e.g., a trie or sorted array, storing for each word a list of its occurrences
- Each word of the text is read sequentially and searched in the vocabulary
- If it is not found, it is added to the vocabulary with an empty list of occurrences
- The word position is added to the end of its list of occurrences

Construction of the Inverted File – Step 2

Step 2: Storage phase (once the text is exhausted)

- The list of occurrences is written contiguously to the disk (posting file)
- The vocabulary is stored in lexicographical order (index file) in main memory together with a pointer for each word to its list in the posting file

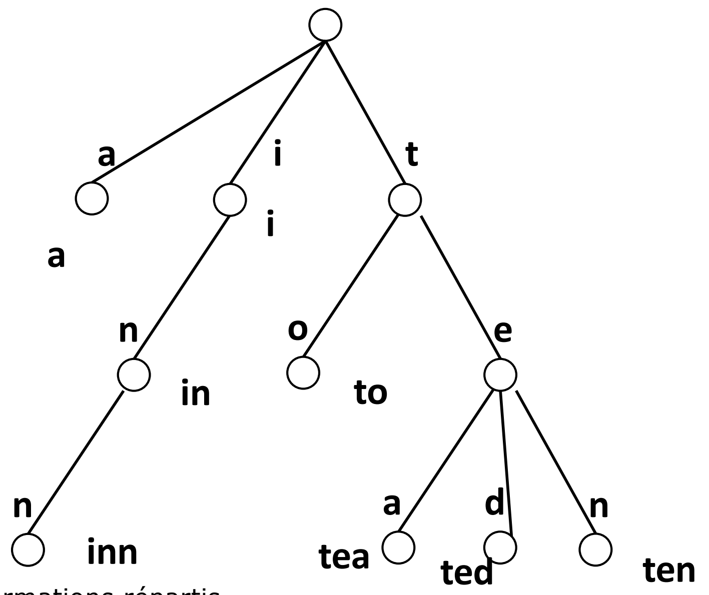
Overall cost O(n)

Tries

A trie is a tree data structure to index strings

- For each prefix of each length in the set of strings a separate path is created
- Strings are looked up by following the prefix path

Example: trie for {a, to, tea, ted, ten, i, in, inn}

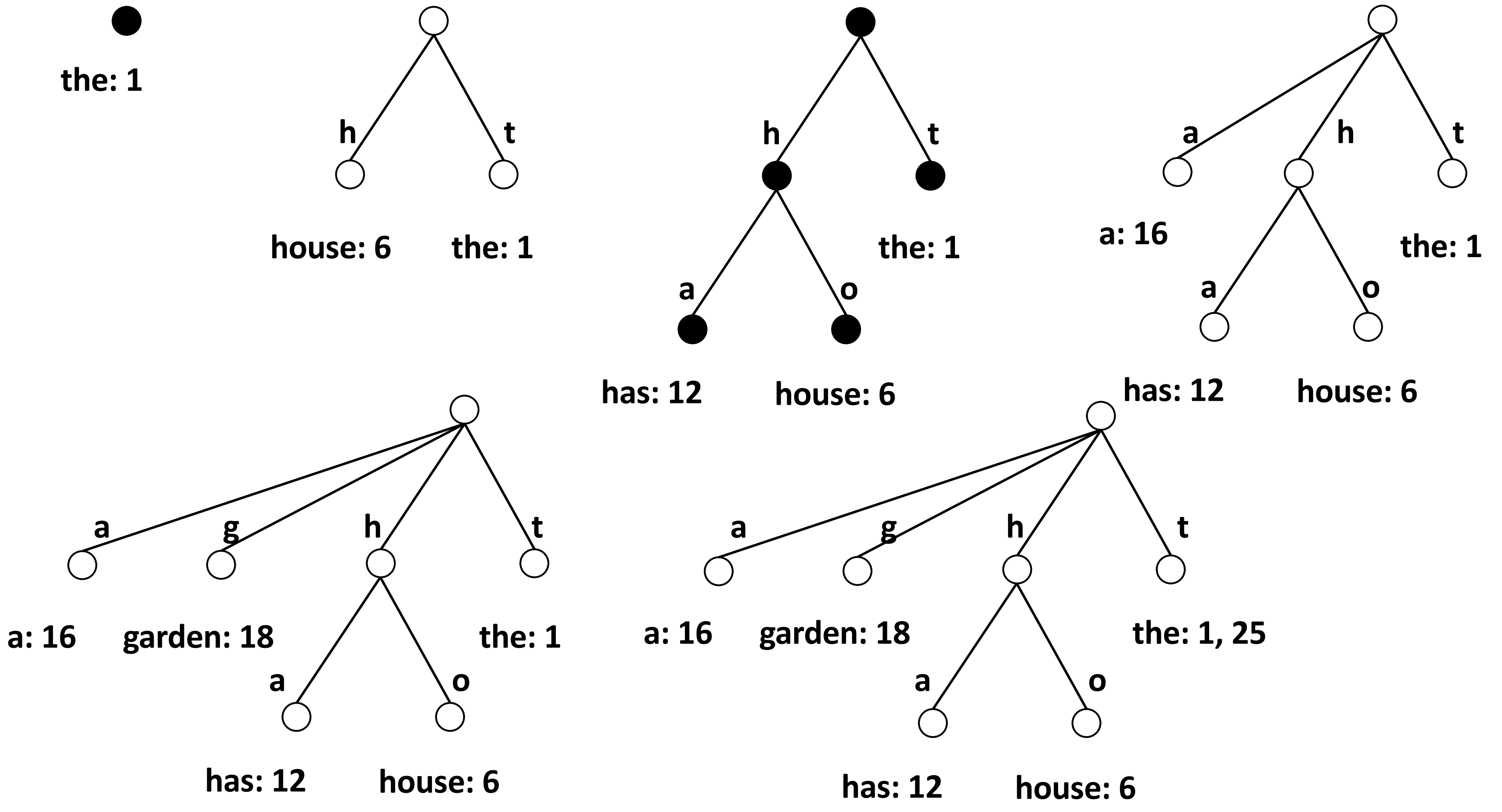


Example

1 6 12 16 18 25 29 36 40 45 54 58 66 70

the house has a garden. the garden has many flowers. the flowers are beautiful

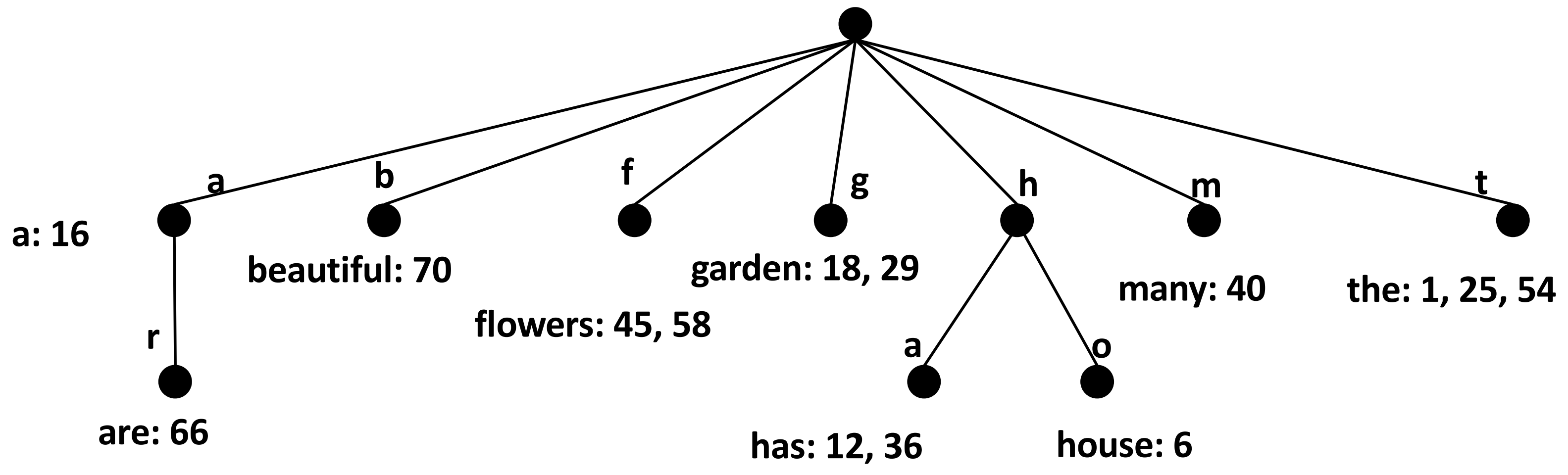
(each word = one document, position = document identifier)



Example

1 6 12 16 18 25 29 36 40 45 54 58 66 70

the house has a garden. the garden has many flowers. the flowers are beautiful



inverted file I

a: 16
are: 66
beautiful: 70
flowers: 45, 58
garden: 18, 29
has: 12, 36
house: 6
many: 40
the: 1, 25, 54

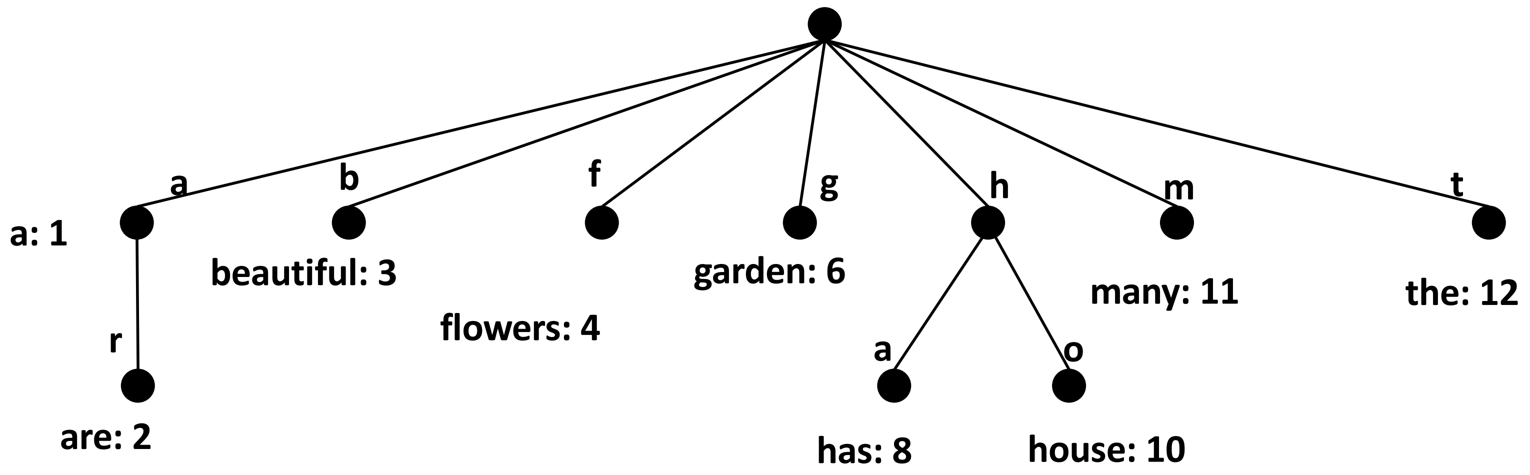
16, 66, 70, 45, 58, 18, 29, 12, 36, 6, 40, 1, 25, 54

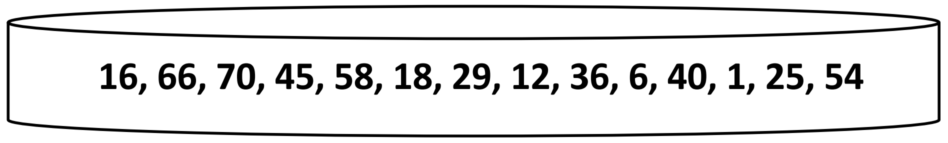
postings file

Example

1 6 12 16 18 25 29 36 40 45 54 58 66 70

the house has a garden. the garden has many flowers. the flowers are beautiful





postings file

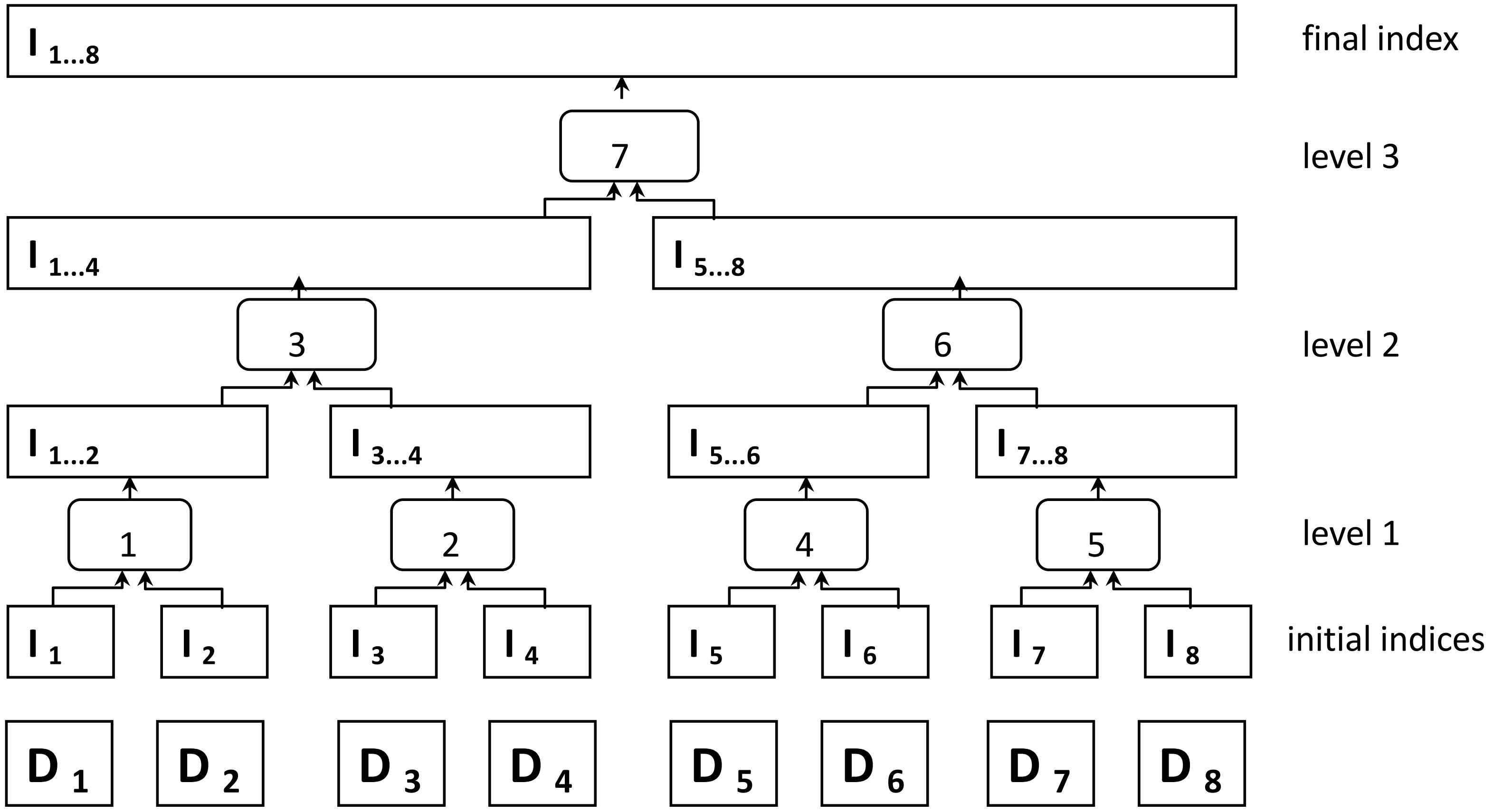
Index Construction in Practice

When using a single node not all index information can be kept in main memory → Index merging

- When no more memory is available, a partial index I_i is written to disk
- The main memory is erased before continuing with the rest of the text
- Once the text is exhausted, a number of partial indices I_i exist on disk
- The partial indices are merged to obtain the final index

Index Merging

BSBI: Blocked sort-based Indexing



Example

```
1    6       12   16  18        25   29       36          40     45        54   58        66  70
the house has a  garden. the garden has       many flowers. the flowers are beautiful
```

inverted file I1

a: 16
garden: 18, 29
has: 12, 36
house: 6
the: 1, 25

are: 66
beautiful: 70
flowers: 45, 58
many: 40
the: 54

inverted file I2

a: 16
are: 66
beautiful: 70
flowers: 45, 58
garden: 18, 29
has: 12, 36
house: 6
many: 40
the: 1, 25, 54

1, 25 + 54 -> 1, 25, 54

concatenate inverted lists

total cost: $O(n \log_2(n/M))$

M size of memory

Addressing Granularity

Documents can be addressed at different granularities

- coarser: text blocks spanning multiple documents
- finer: paragraph, sentence, word level

General rule

- the finer the granularity the less post-processing but the larger the index

Example: index size in % of document collection size

Index	**Small collection (1Mb)**	**Medium collection (200Mb)**	**Large collection (2Gb)**
Addressing words	**73%**	**64%**	**63%**
Addressing documents	**26%**	**32%**	**47%**
Addressing 256K blocks	**25%**	**2.4%**	**0.7%**

Index Compression

Documents are ordered and each document identifier d_{ij} is replaced by the difference to the preceding document identifier

- Document identifiers are encoded using fewer bits for smaller, common numbers

$$l_k = \left\langle f_k : d_{i_1}, \dots, d_{i_{fk}} \right\rangle \rightarrow$$

$$l_k' = \left\langle f_k : d_{i_1}, d_{i_2} - d_{i_1}, \dots, d_{i_{fk}} - d_{i_{fk}-1} \right\rangle$$

- Use of varying length compression further reduces space requirement
- In practice index is reduced to 10- 15% of database size

X	code(X)
1	0
2	10 0
3	10 1
4	110 00
5	110 01
6	110 10
7	110 11
8	1110 000
63	111110 11111

Using a trie in index construction …

1. Helps to quickly find words that have been seen before
2. Helps to quickly decide whether a word has not seen before
3. Helps to maintain the lexicographic order of words seen in the documents
4. All of the above

Web-Scale Index Construction: Map-Reduce

Pioneered by Google: 20PB of data per day (2008)

- Scan 100 TB on 1 node @ 50 MB/s = 23 days
- Scan on 1000-node cluster = 33 minutes

Cost-efficiency

- Commodity nodes, network (cheap, but unreliable)
- Automatic fault-tolerance (fewer admins)
- Easy to use (fewer programmers)

Map-Reduce Programming Model

Data type: key-value pairs (k, v)

Map function: $(k_{in}, v_{in}) \rightarrow [(k_{inter}, v_{inter})]$

Analyses some input, and produces a list of results

Reduce function: $(k_{inter}, [v_{inter}]) \rightarrow [(k_{out}, v_{out})]$

Takes all results belonging to one key, and computes aggregates

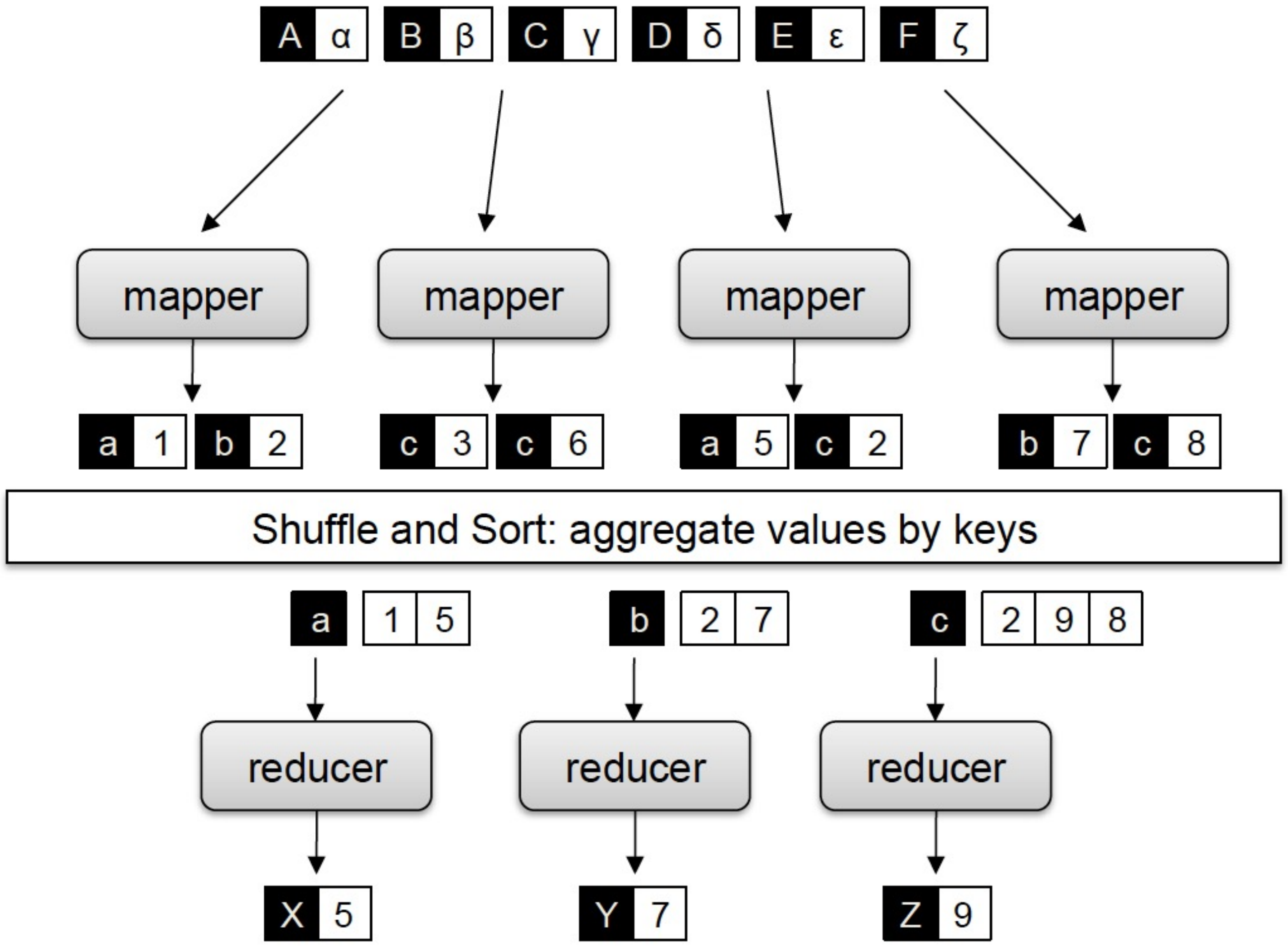
Example

Basic word counter program

```
def mapper(document, line):
  for word in line.split(): output(word, 1)

def reducer(key, values):
  output(key, sum(values))
```

Map-Reduce Processing Model



The input data is partitioned into subsets

Mappers extract word occurrences

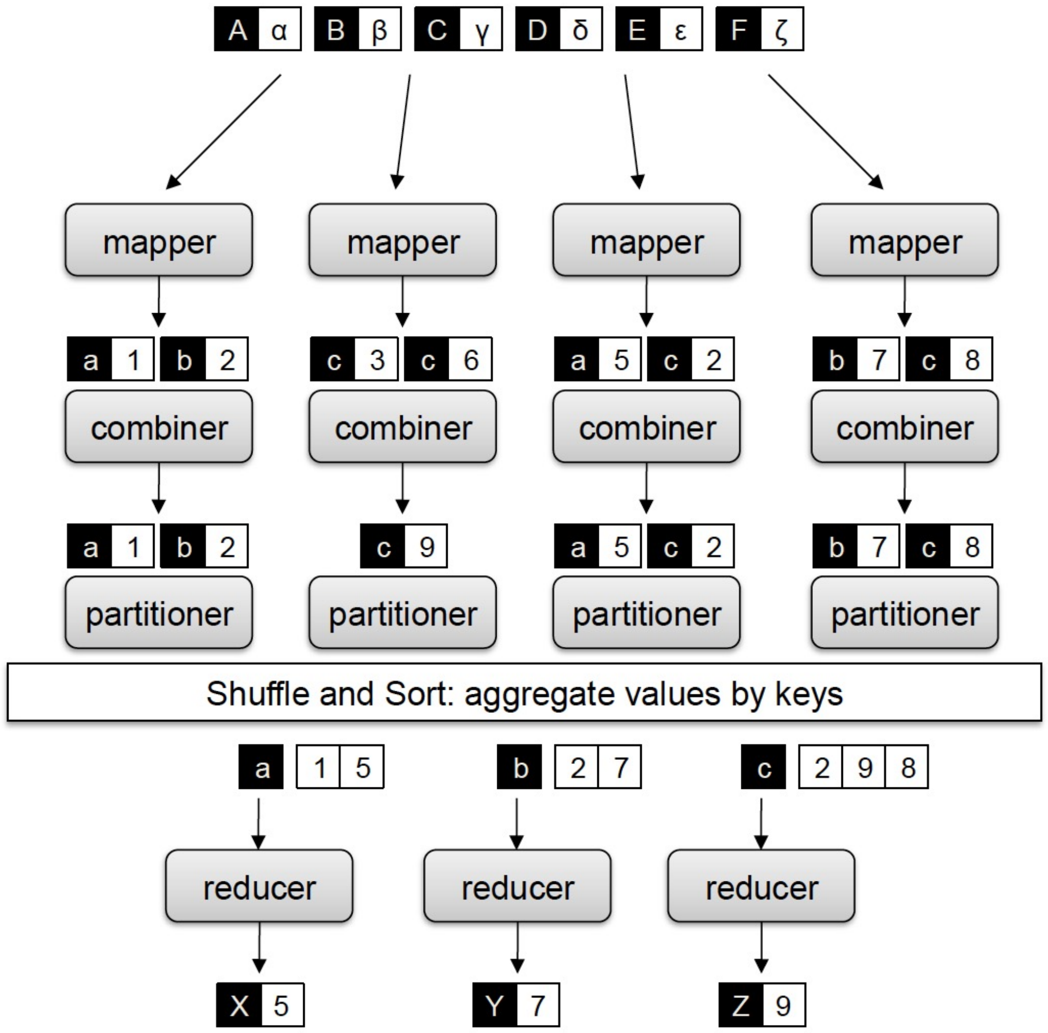
The assigned reduce process is chosen

Reducers aggregate word occurrences

Output is written to stable storage

Important: the reducers can only start after all mappers have finished!

Refined Map-Reduce Programming Model



Combiners work like reducers, but only on the local data of a mapper

```
def combiner(key, values):
  output(key, sum(values))
```

Partitioners allow to control the strategy for distributing keys to reducers

What the Programmer Controls (and not)

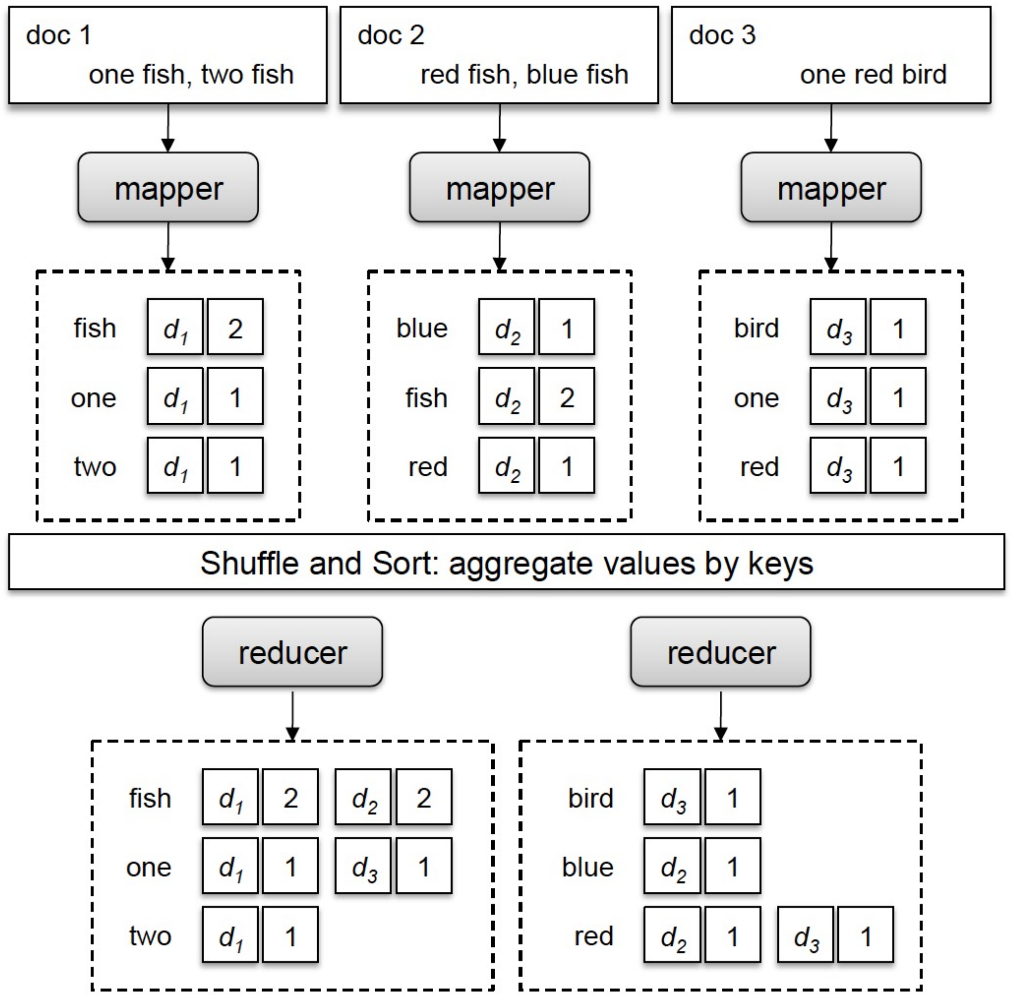
The programmer controls

- Key-value data structures (can be complex)
- Maintenance of state in mappers and reducers
- Sort order of intermediate key-value pairs
- Partitioning scheme on the key space

The map-reduce platform controls

- where the mappers and reducers run
- when a mapper and reducer starts and terminates
- which input data is assigned to a specific mapper
- which intermediate key-value pairs are processed by a specific reducer

Inverted File Construction Using Map-Reduce



Mappers extract postings from document

Postings are provided to reducers

Reducers aggregate posting lists

Inverted File Construction Program

```
def mapper(document, text):
     f = {}
     for word in text.split(): f[word] += 1
     for word in f.keys():
          output(word, (document, f[word]))

def reducer(key, postings):
     p = []
     for d, f in postings: p.append((d, f))
     p.sort()
     output(key, p)
```

Other Applications of Map-Reduce

Framework is used in many other tasks, particular for text and Web data processing

- Graph processing (e.g. PageRank)
- Processing relational joins
- Learning probabilistic models

Maintaining the order of document identifiers for vocabulary construction when partitioning the document collection is important ...

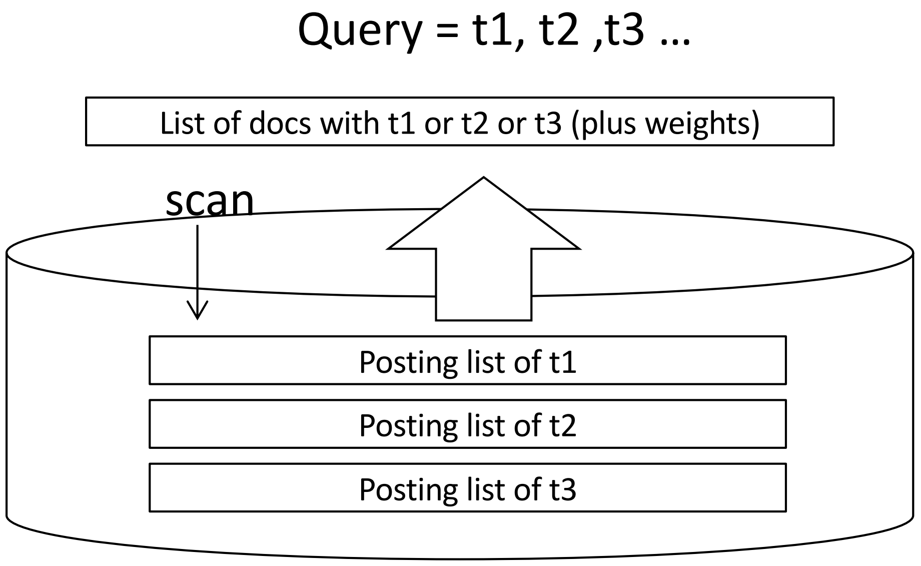
1. in the index merging approach for single node machines
2. in the map-reduce approach for parallel clusters
3. in both
4. in neither of the two

8. DISTRIBUTED RETRIEVAL

Retrieval Processing

Centralized retrieval

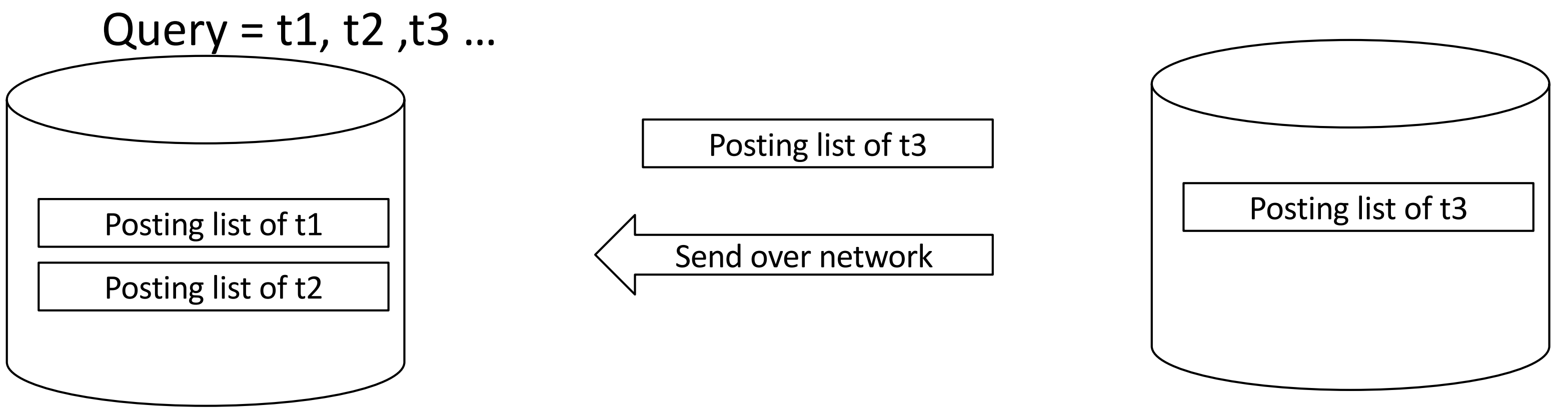
- Aggregate the weights for ALL documents by scanning the posting lists of the query terms
- Scanning is relatively efficient
- Computationally quite expensive (memory, processing)



Distributed Retrieval

Distributed retrieval

- Posting lists for different terms stored on different nodes
- The transfer of complete posting lists can become prohibitively expensive in terms of bandwidth consumption



Is it necessary to transfer the complete posting list to identify the top-k documents?

Fagin’s Algorithm

Entries in posting lists are sorted according to the tf-idf weights

- Scan in parallel all lists in round-robin till k documents are detected that occur in all lists
- Lookup the missing weights for documents that have not been seen in all lists
- Select the top-k elements

Algorithm provably returns the top-k documents for monotone aggregation functions!

Example 1

Finding the top-2 elements for a two-term query

d1	0.9
d4	0.82
d3	0.8
d5	0.65
…..	
d6	0.51
d2	0.1
d7	0.0

d6	0.81
d2	0.7
d5	0.66
d1	0.45
…..	
d3	0.33
d7	0.15
d4	0.0

Example 2

Finding the top-2 elements for a two-term query

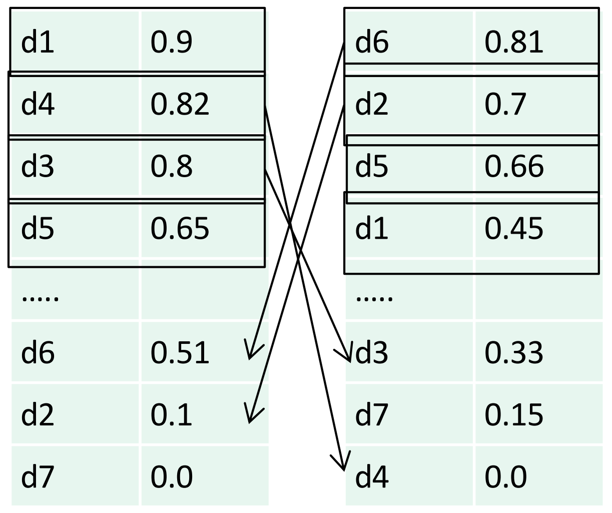
d1	0.9
d4	0.82
d3	0.8
d5	0.65
…..	
d6	0.51
d2	0.1
d7	0.0

d6	0.81
d2	0.7
d5	0.66
d1	0.45
…..	
d3	0.33
d7	0.15
d4	0.0

d1	0.9	0.45	1.35
d6		0.81	0.81
d4	0.82		0.82
d2		0.7	0.7
d3	0.8		0.8
d5	0.65	0.66	1.34

Example 3

Finding the top-2 elements for a two-term query



d1	0.9
d4	0.82
d3	0.8
d5	0.65
…..	
d6	0.51
d2	0.1
d7	0.0

d6	0.81
d2	0.7
d5	0.66
d1	0.45
…..	
d3	0.33
d7	0.15
d4	0.0

d1	0.9	0.45	1.35
d6	0.51	0.81	1.32
d4	0.82	0.0	0.82
d2	0.1	0.7	0.8
d3	0.8	0.33	1.13
d5	0.65	0.66	1.31

Discussion

Complexity

- $O((k\ n)^{1/2})$ entries are read in each list for n documents
- Assuming that entries are uncorrelated
- Improves if they are positively correlated

In distributed settings optimizations to reduce the number of roundtrips

- Send a longer prefix of one list to the other node

Fagin’s algorithm may behave poorly in practical cases

- Alternative algorithm: Threshold Algorithm

Threshold Algorithm

Threshold Algorithm

- Access sequentially elements in each list
- At each round
    - lookup missing weights of current elements in other lists using random access
    - Keep the top k elements seen so far
    - Compute threshold as aggregate value of elements seen in current round
- If at least k documents have aggregate value higher than threshold, halt

Example

Finding the top-2 elements for a two-term query

d1	0.9
d4	0.82
d3	0.8
d5	0.65
…..	
d6	0.51
d2	0.1
d7	0.0

d6	0.81
d2	0.7
d5	0.66
d1	0.45
…..	
d3	0.33
d7	0.15
d4	0.0

Threshold

1.71

d1	0.9	0.45	**1.35**
d6	0.81	0.51	**1.32**

d1 and d6 have aggregate weights lower than the threshold, therefore continue

Example

Threshold

d1	0.9
d4	0.82
d3	0.8
d5	0.65
…..	
d6	0.51
d2	0.1
d7	0.0

d6	0.81
d2	0.7
d5	0.66
d1	0.45
…..	
d3	0.33
d7	0.15
d4	0.0

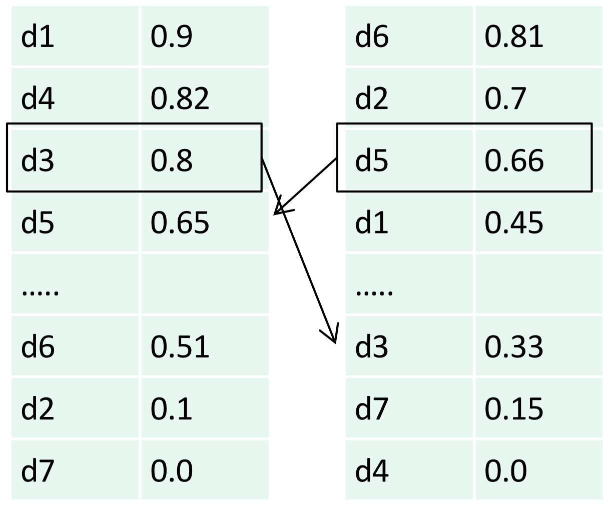
1.71
1.52

d1	0.9	0.45	**1.35**
d6	0.81	0.51	**1.32**

The documents d4, d2 have lower aggregate weights and are therefore dismissed

d1 and d6 have aggregate weights lower than the threshold, therefore continue

Example



d1	0.9
d4	0.82
d3	0.8
d5	0.65
…..	
d6	0.51
d2	0.1
d7	0.0

d6	0.81
d2	0.7
d5	0.66
d1	0.45
…..	
d3	0.33
d7	0.15
d4	0.0

Threshold

1.71
1.52
1.46

d1	0.9	0.45	**1.35**
d6	0.81	0.51	**1.32**

The documents d3, d5 have lower aggregate weights and are therefore dismissed

d1 and d6 have aggregate weights lower than the threshold, therefore continue

Example

d1	0.9
d4	0.82
d3	0.8
$\boxed{\text{d5} \quad 0.65}$	
…..	
d6	0.51
d2	0.1
d7	0.0

d6	0.81
d2	0.7
d5	0.66
$\boxed{\text{d1} \quad 0.45}$	
…..	
d3	0.33
d7	0.15
d4	0.0

Threshold

1.71
1.52
1.46
1.1

d1	0.9	0.45	**1.35**
d6	0.81	0.51	**1.32**

The document d5 has lower aggregate weights and is therefore dismissed

In this round d1 and d6 have aggregate weights higher than the threshold and the algorithm terminates

Discussion

TA in general performs fewer rounds the FA

- Therefore fewer document accesses
- But more random accesses

TA is also provably correct for monotone aggregation functions

Applications

Beyond distributed document retrieval these algorithms have wider applications

- Multimedia, image retrieval
- Top-k processing in relational databases
- Document filtering
- Sensor data processing

When applying Fagin's algorithm for a query with three different terms for finding the k top documents, the algorithm will scan ...

1. 2 different lists
2. 3 different lists
3. k different lists
4. it depends how many rounds are taken

With Fagin’s algorithm, once k documents have been identified that occur in all of the lists ...

1. These are the top-k documents
2. The top-k documents are among the documents seen so far
3. The search has to continue in round-robin till the top-k documents are identified
4. Other documents have to be searched to complete the top-k list