Vector Space Model Implementation

Presenter

Quách Đình Hoàng

Slides are obtained from ChengXiang Zhai and Sean Massung book

Outline

1. Text Retrieval Problem
2. Vector Space Model
3. Vector Space Model Implementation
   - Term At A Time (TAAT) Query Processing
   - Document At A Time (DAAT) Query Processing

1. Text Retrieval Problem

Text Retrieval Problem

- **Query:** $q = q_1,\ldots,q_m$, where $q_i \in V$
- **Document:** $d = d_1,\ldots,d_n$, where $d_i \in V$
- **Ranking function:** $f(q, d) \in \Re$
- A good ranking function should rank relevant documents on top of non-relevant ones
- **Key challenge**: how to measure the likelihood that document d is relevant to query q
- **Retrieval model** = formalization of relevance (give a computational definition of relevance)

Common Form of a Retrieval Function

$f(q =$ "presidential campaign news", $\boxed{d})$

"Bag of Words"

$g($"presidential", $\boxed{d})$ $g($"campaign", $\boxed{d})$ $g($"news", $\boxed{d})$

How many times does "presidential" occur in d?
Term frequency (TF): $c($"presidential", $d)$

How long is d? **Document length:** $|d|$

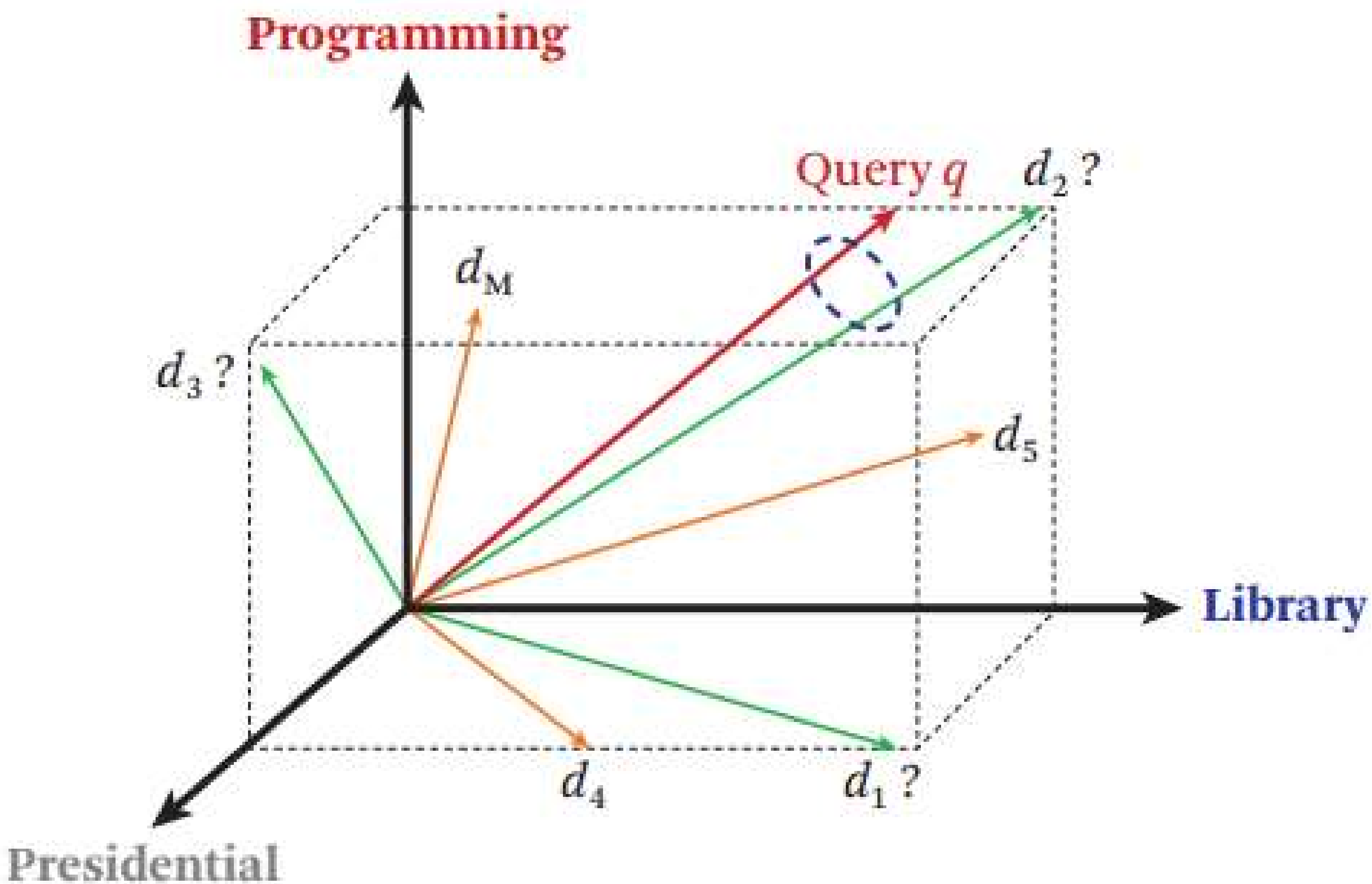
How often do we see "presidential" in the entire collection?
Document frequency: DF("presidential")
P("presidential"|collection)

2. Vector Space Model

Vector Space Model (VSM)

- Represent a document/query by a ***term vector***
  - ***Term***: basic concept, e.g., ***word*** or ***phrase***
  - Each term defines one dimension
  - N terms define a high-dimensional space
  - Element of vector corresponds to term weight
  - E.g., $d = (x_1,\ldots,x_N)$, x_i is "importance" of term i
- Measure relevance by the distance between the query vector and document vector in the vector space

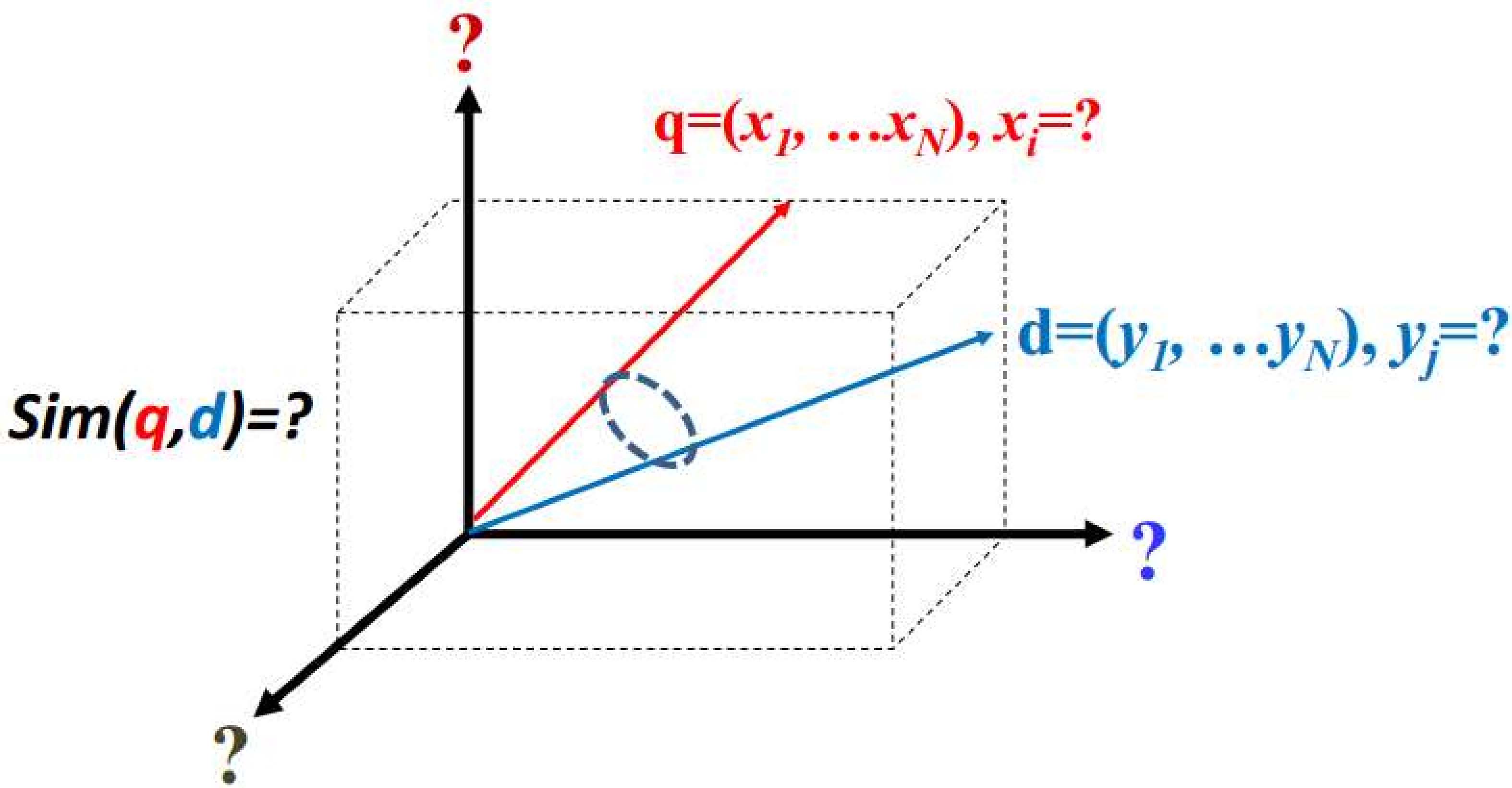
Vector Space Model (VSM) Illustration



VSM is a Framework

- How to define/select the terms
  - Terms are assumed to be linearly independent
- How to assign term weights
  - Weight in query indicates importance of term
  - Weight in doc indicates how well the term characterizes the doc
- How to define the similarity/distance measure

What VSM Doesn't Say



Term weighting

- Major term weighting heuristics
  - TF weighting and transformation
  - IDF weighting
  - Document length normalization

Term frequency		Document frequency		Normalization	
n (natural)	$\mathrm{tf}_{t,d}$	n (no)	1	n (none)	1
l (logarithm)	$1+\log(\mathrm{tf}_{t,d})$	t (idf)	$\log\frac{N}{\mathrm{df}_t}$	c (cosine)	$\frac{1}{\sqrt{w_1^2+w_2^2+\ldots+w_M^2}}$
a (augmented)	$0.5+\frac{0.5\times\mathrm{tf}_{t,d}}{\max_t(\mathrm{tf}_{t,d})}$	p (prob idf)	$\max\{0,\log\frac{N-\mathrm{df}_t}{\mathrm{df}_t}\}$	u (pivoted unique)	$1/u$ (Section 6.4.4)
b (boolean)	$\begin{cases}1 & \text{if } \mathrm{tf}_{t,d}>0\\ 0 & \text{otherwise}\end{cases}$			b (byte size)	$1/\mathit{CharLength}^{\alpha}$, $\alpha<1$
L (log ave)	$\frac{1+\log(\mathrm{tf}_{t,d})}{1+\log(\mathrm{ave}_{t\in d}(\mathrm{tf}_{t,d}))}$				

https://nlp.stanford.edu/IR-book/html/htmledition/document-and-query-weighting-schemes-1.html

State of the Art VSM Ranking Functions

Pivoted length normalization VSM

$$f(q,d) = \sum_{w \in q \cap d} c(w,q) \frac{\ln(1+\ln(1+c(w,d)))}{1-b+b\frac{|d|}{avdl}} \log \frac{M+1}{\mathrm{df}(w)} \quad b \in [0,1]$$

BM25/Okapi

$$f(q,d) = \sum_{w \in q \cap d} c(w,q) \frac{(k+1)c(w,d)}{c(w,d)+k(1-b+b\frac{|d|}{avdl})} \log \frac{M+1}{\mathrm{df}(w)} \quad b \in [0,1],\ k \in [0,+\infty)$$

3. Vector Space Model Implementation

Text Retrieval System Implementation

- Tokenizer
  - This determines how we represent a document
- Indexer
  - This convert documents to data structures that enable fast search
  - Compression when appropriate
- Scorer
  - This use inverted index for fast search

Inverted Index

- Fast access to all docs containing a given term (along with freq and pos information)
- For each term, we get a list of tuples (docID, freq, pos).
- Given a query, we can fetch the lists for all query terms and work on the involved documents.
  – Boolean query: set operation
  – Natural language query: term weight summing
- More efficient than scanning docs

Inverted Index Example

Doc 1

This is a sample document
with one sample sentence

Doc 2

This is another sample document

Dictionary

Term	# docs	Total freq
This	2	2
is	2	2
sample	2	3
another	1	1
…	…	…

Postings

Doc id	Freq
1	1
2	1
1	1
2	1
1	2
2	1
2	1
…	…

term|#docs|totalfreq|docID1:freq1;docID2:freq2;…

Positional Inverted Index Example

Doc 1

web retrieval
web search
information

Doc 2

search engine
web ranking

Doc 3

web search
course
information
search

Dictionary

Term	# docs	Total freq
course	1	1
engine	2	1
information	2	2
ranking	1	1
retrieval	1	1
search	3	4
web	3	4
…	…	…

Postings

Doc id	Freq	Pos
3	1	2
2	1	1
1	1	4
3	1	3
2	1	3
2	1	1
1	1	3
2	1	0
3	2	1, 4
1	2	0, 2
2	1	2
3	1	1
…	…	…

term|#docs|totalfreq|docID1:freq1,pos1,pos2;docID2:freq2,pos3,pos4,pos5;…

How to Score Documents Quickly with Inverted Index

General Form of Scoring Function

Final score **adjustment**

$$f(q,d) = f_a(\boldsymbol{h}(\boldsymbol{g}(\boldsymbol{t_1},\boldsymbol{d},\boldsymbol{q}),\ldots,\boldsymbol{g}(\boldsymbol{t_k},\boldsymbol{d},\boldsymbol{q})),f_d(d),f_q(q))$$

Weight **aggregation**

Weight a **matched** query term in d

TAAT vs DAAT Query Processing

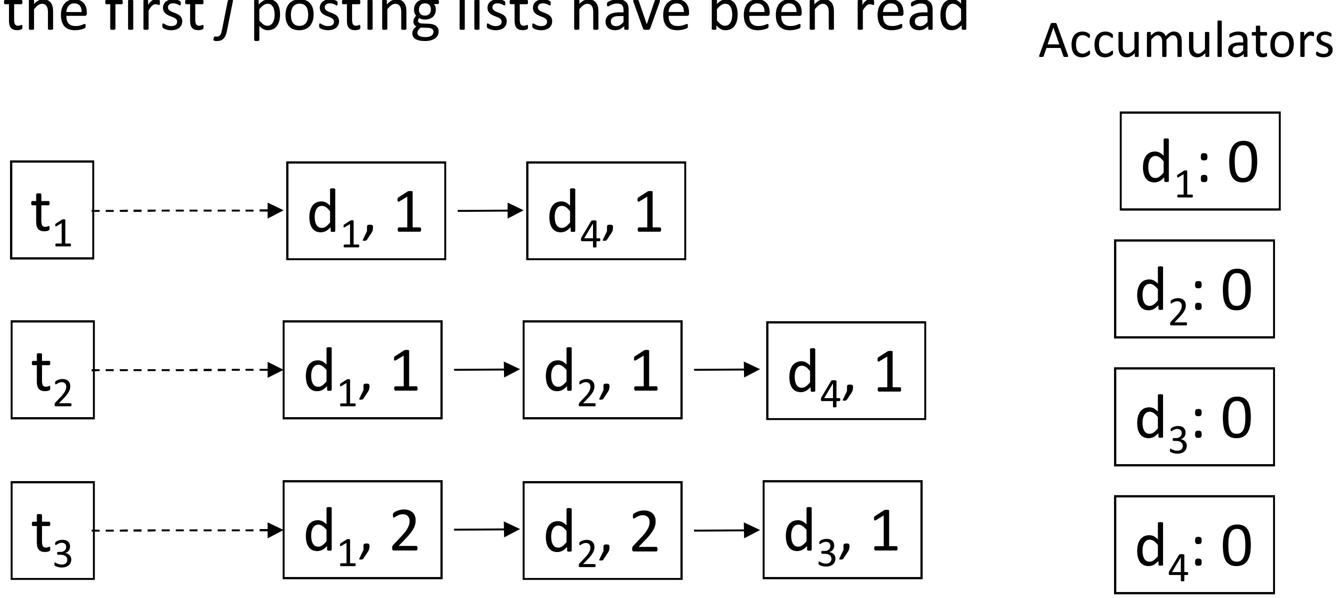
- **TAAT = "Term At A Time"**
  - Scores for all docs computed concurrently, one query term at a time
- **DAAT = "Document At A Time"**
  - Total score for each doc (include all query terms) computed, before proceeding to the next
- Each has implications for how the retrieval index is structured and stored

Term-at-a-time Ranking

- Read posting lists for query terms ($t_1, \ldots, t_{|q|}$) successively
- Maintains an accumulator for each result document with value

$$acc(d) = \sum_{i \leq j} score(t_i, d)$$

after the first *j* posting lists have been read

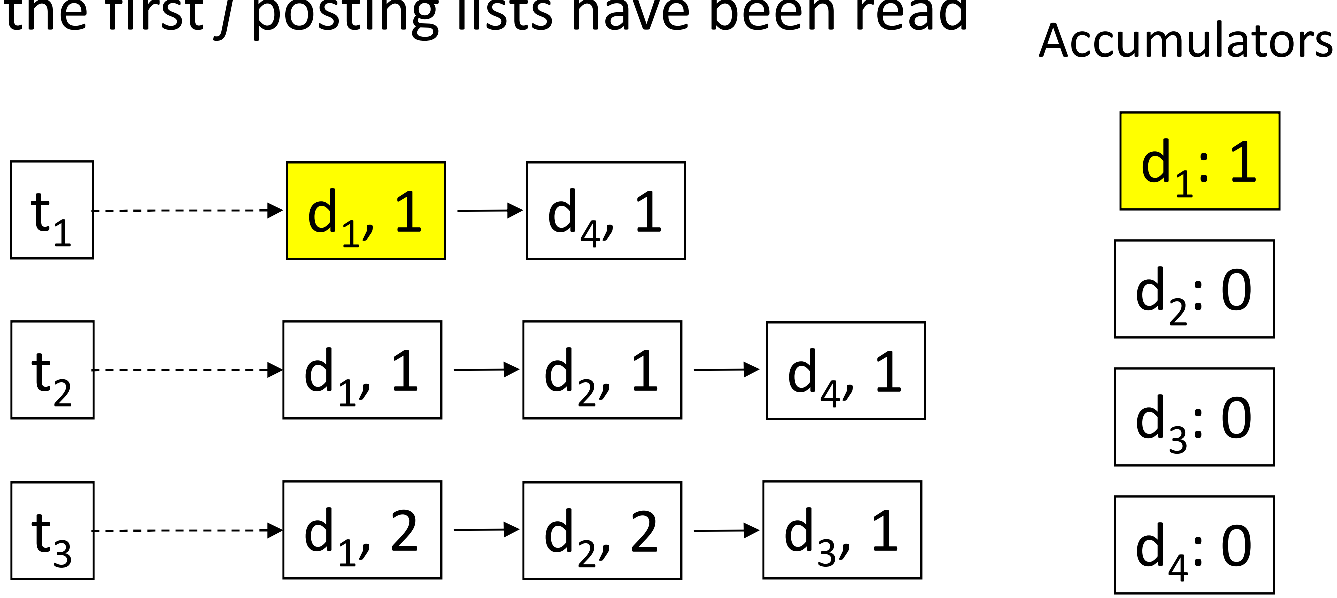


Term-at-a-time Ranking

- Read posting lists for query terms (t_1, …, $t_{|q|}$) successively
- Maintains an accumulator for each result document with value

$$acc(d) = \sum_{i \leq j} score(t_i, d)$$

after the first *j* posting lists have been read

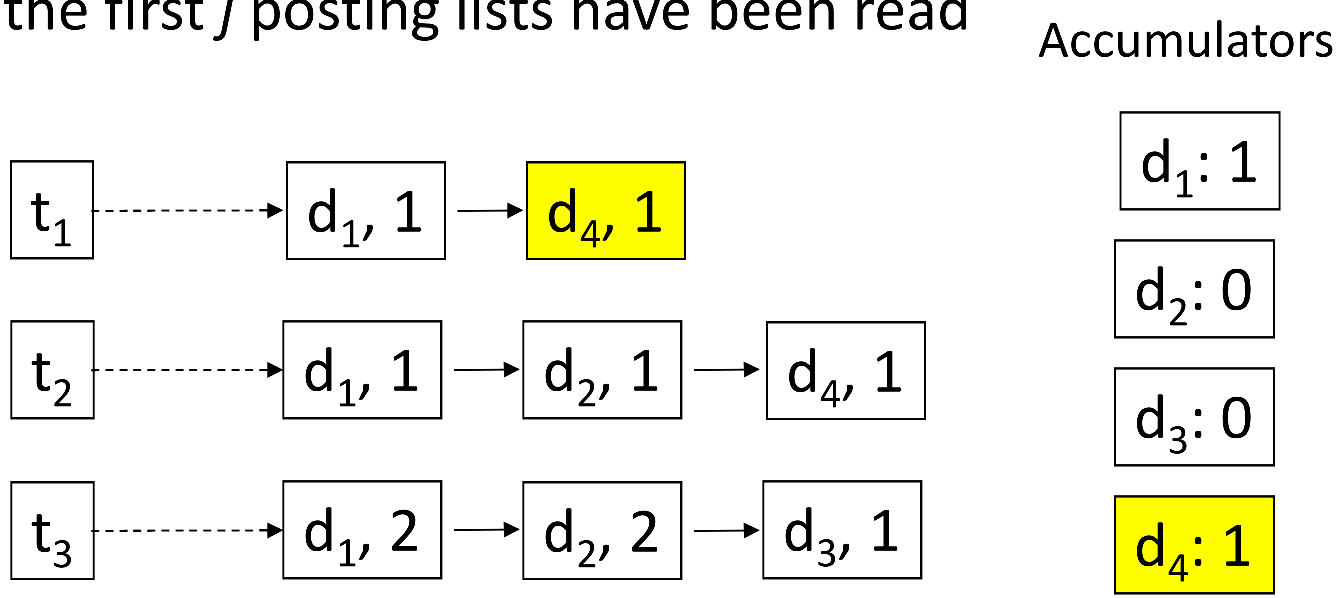


Term-at-a-time Ranking

- Read posting lists for query terms (t_1, …, $t_{|q|}$) successively
- Maintains an accumulator for each result document with value

$$acc(d) = \sum_{i \le j} score(t_i, d)$$

after the first *j* posting lists have been read



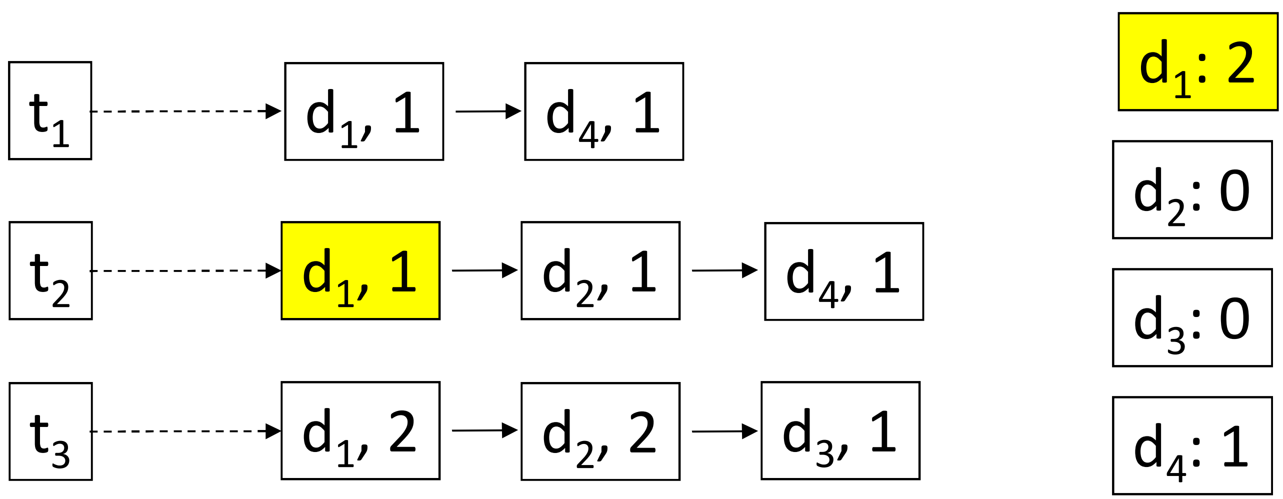
Term-at-a-time Ranking

- Read posting lists for query terms (t_1, …, $t_{|q|}$) successively
- Maintains an accumulator for each result document with value

$$acc(d) = \sum_{i \leq j} score(t_i, d)$$

after the first *j* posting lists have been read

Accumulators

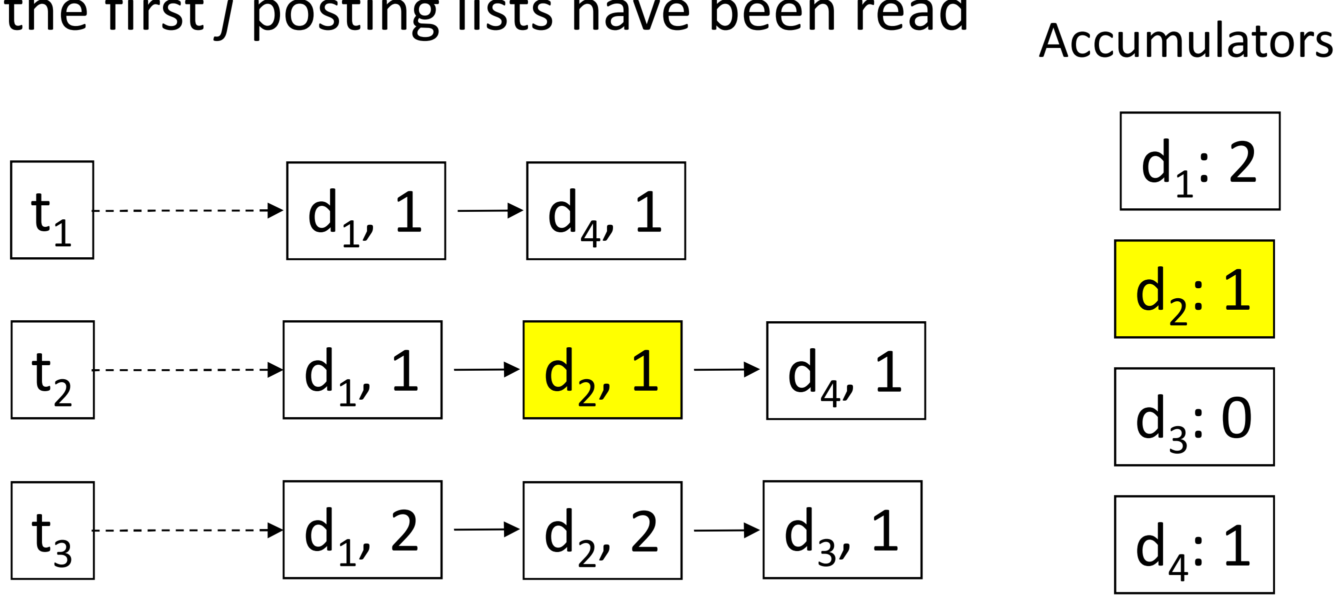


Term-at-a-time Ranking

- Read posting lists for query terms ($t_1, \ldots, t_{|q|}$) successively
- Maintains an accumulator for each result document with value

$$acc(d) = \sum_{i \leq j} score(t_i, d)$$

after the first *j* posting lists have been read

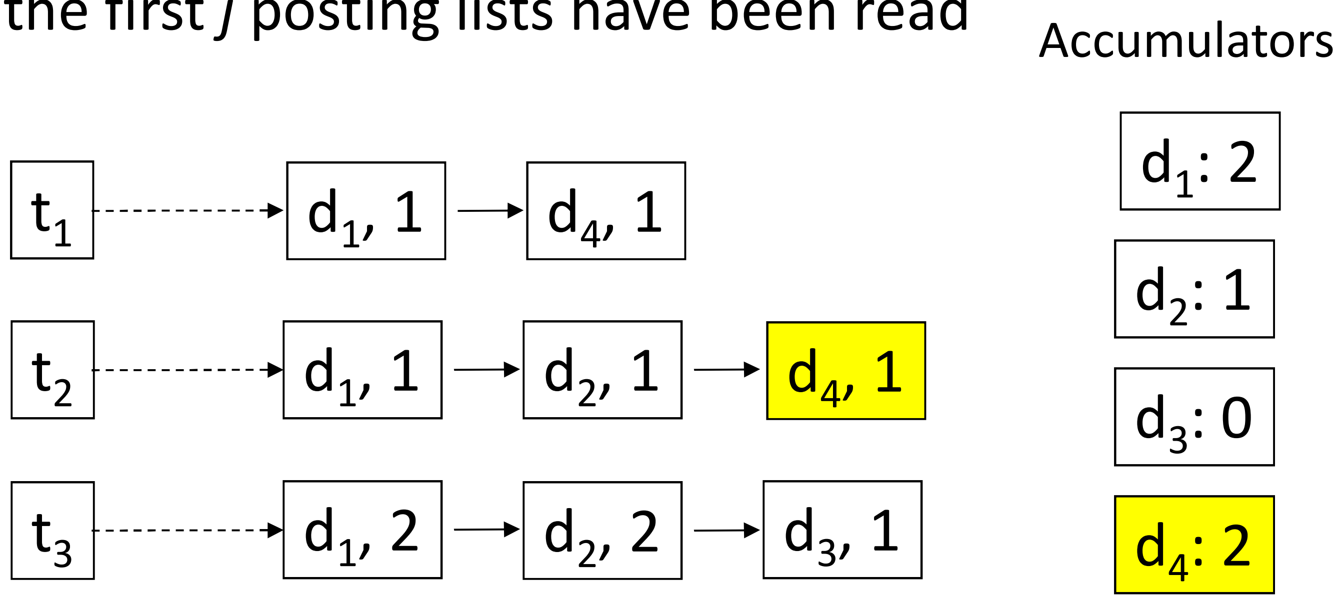


Term-at-a-time Ranking

- Read posting lists for query terms ($t_1, \ldots, t_{|q|}$) successively
- Maintains an accumulator for each result document with value

$$acc(d) = \sum_{i \leq j} score(t_i, d)$$

after the first *j* posting lists have been read

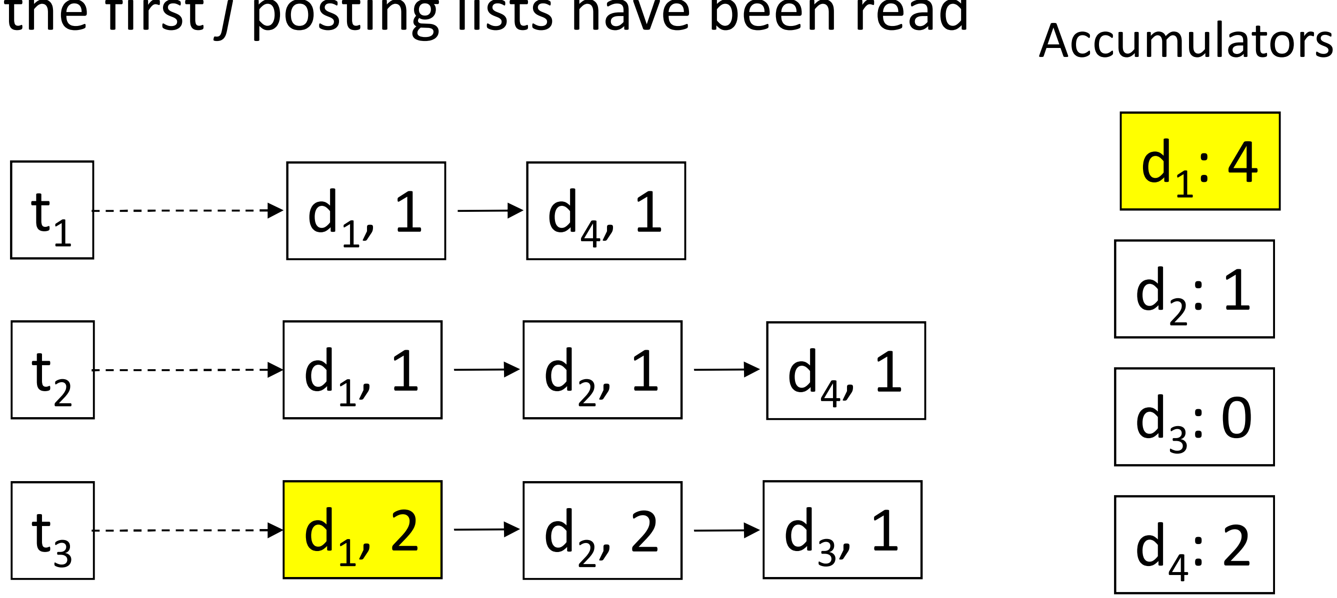


Term-at-a-time Ranking

- Read posting lists for query terms (t_1, …, $t_{|q|}$) successively
- Maintains an accumulator for each result document with value

$$acc(d) = \sum_{i \leq j} score(t_i, d)$$

after the first *j* posting lists have been read

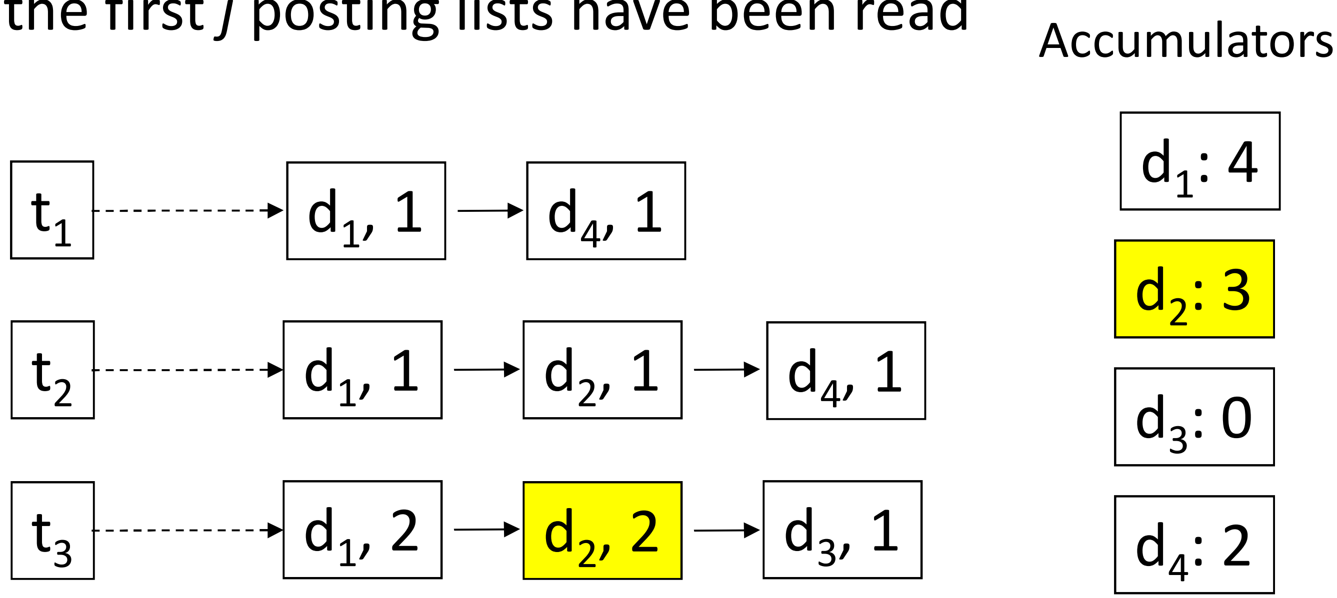


Term-at-a-time Ranking

- Read posting lists for query terms (t_1, …, $t_{|q|}$) successively
- Maintains an accumulator for each result document with value

$$acc(d) = \sum_{i \leq j} score(t_i, d)$$

after the first *j* posting lists have been read

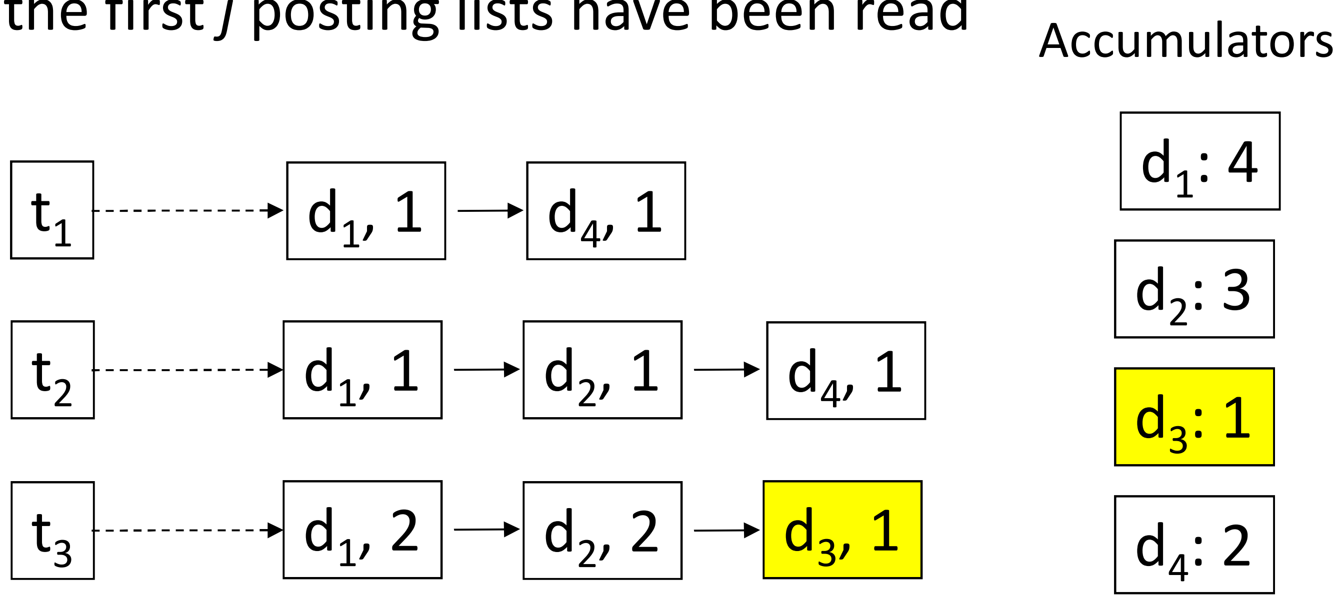


Term-at-a-time Ranking

- Read posting lists for query terms (t_1, …, $t_{|q|}$) successively
- Maintains an accumulator for each result document with value

$$acc(d) = \sum_{i \leq j} score(t_i, d)$$

after the first *j* posting lists have been read



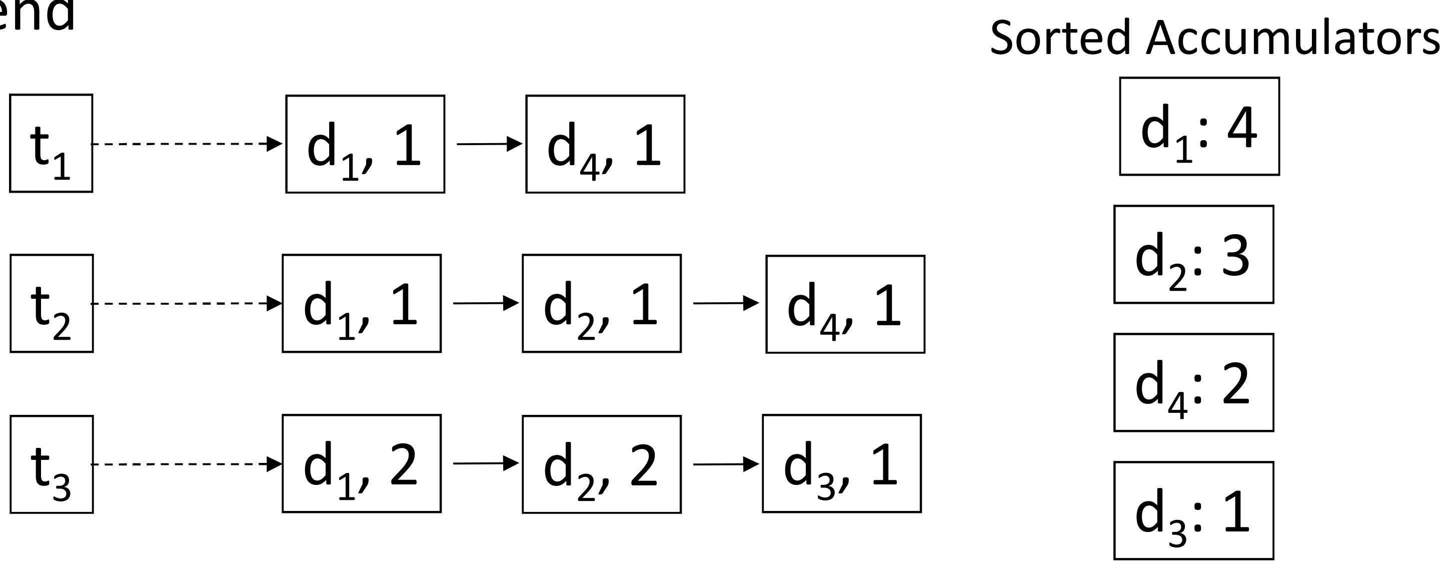
Term-at-a-time Ranking

- Read posting lists for query terms (t_1, …, $t_{|q|}$) successively
- Maintains an accumulator for each result document with value

$$acc(d) = \sum_{i \leq j} score(t_i, d)$$

after the first *j* posting lists have been read

- Top-*k* results can be determined by sorting accumulators at the end



Term-at-a-time Ranking

```
scores = {}        // score accumulator maps doc IDs to scores
for w ∈ q do
    for d, count ∈ Idx.fetch_docs(w) do
        scores[d] = scores[d] + score_term(count)
    end for
end for
return top k documents from scores
```

Disadvantage of term-at-a-time ranking

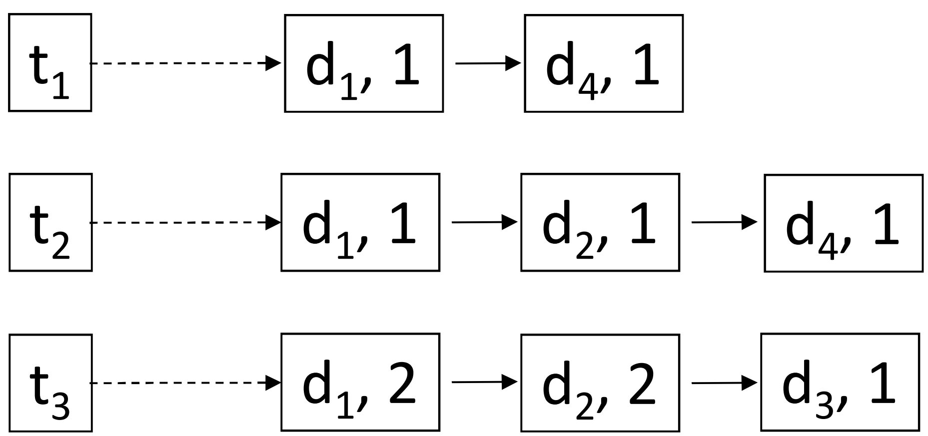
- The size of the score accumulators *scores* will be the size of the number of documents matching at least one term.
- This set is still large.

Document-at-a-time Ranking

- Since most searches are top-*k* searches, we can only keep the top-*k* documents at any one time.
- We can hold the *k* best completely scored documents with a priority queue.

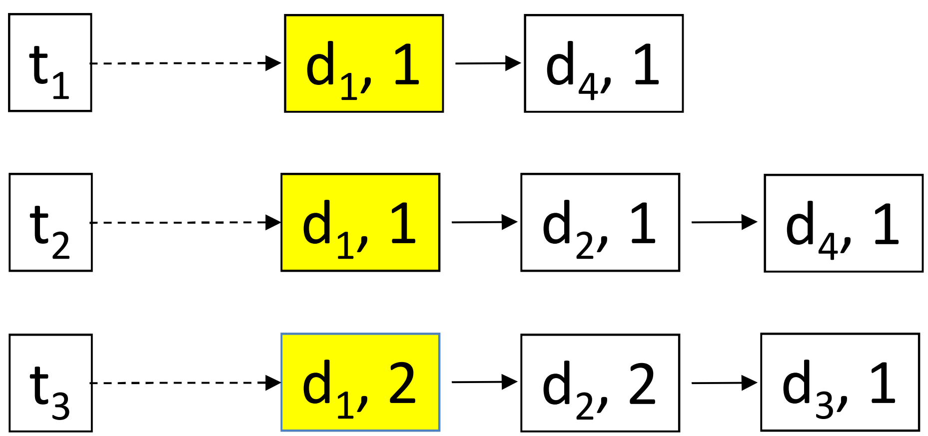
Document-at-a-time Ranking

- Read posting lists for query terms (t_1, …, $t_{|q|}$) concurrently
- Computes score when same document is seen in one or more posting lists
- Always advances posting list with lowest current document id
- Top-k results can be determined by keeping results in priority queue



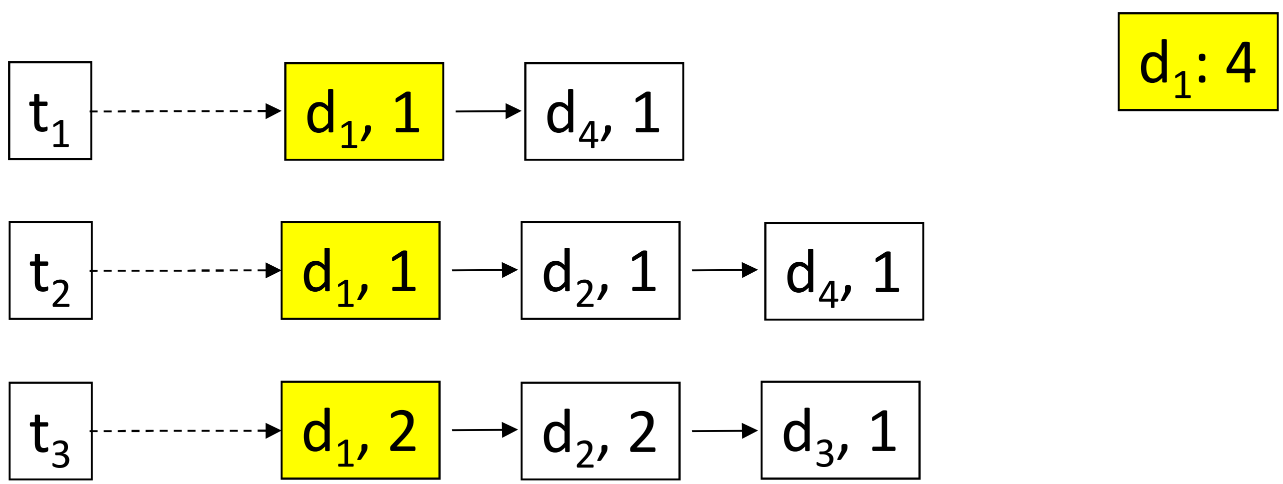
Document-at-a-time Ranking

- Read posting lists for query terms (t_1, …, $t_{|q|}$) concurrently
- Computes score when same document is seen in one or more posting lists
- Always advances posting list with lowest current document id
- Top-k results can be determined by keeping results in priority queue



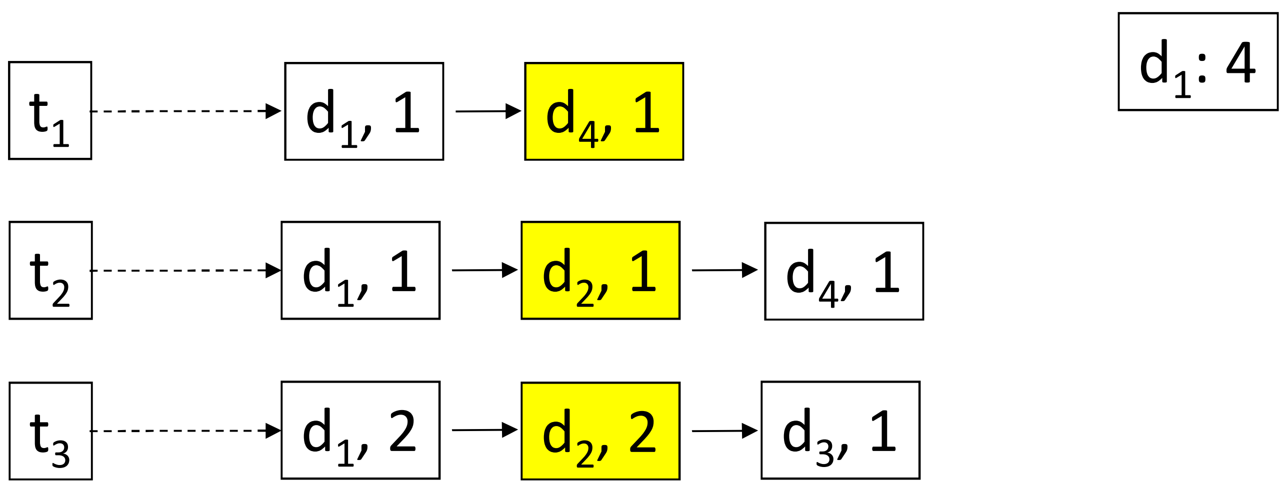
Document-at-a-time Ranking

- Read posting lists for query terms (t_1, …, $t_{|q|}$) concurrently
- Computes score when same document is seen in one or more posting lists
- Always advances posting list with lowest current document id
- Top-k results can be determined by keeping results in priority queue



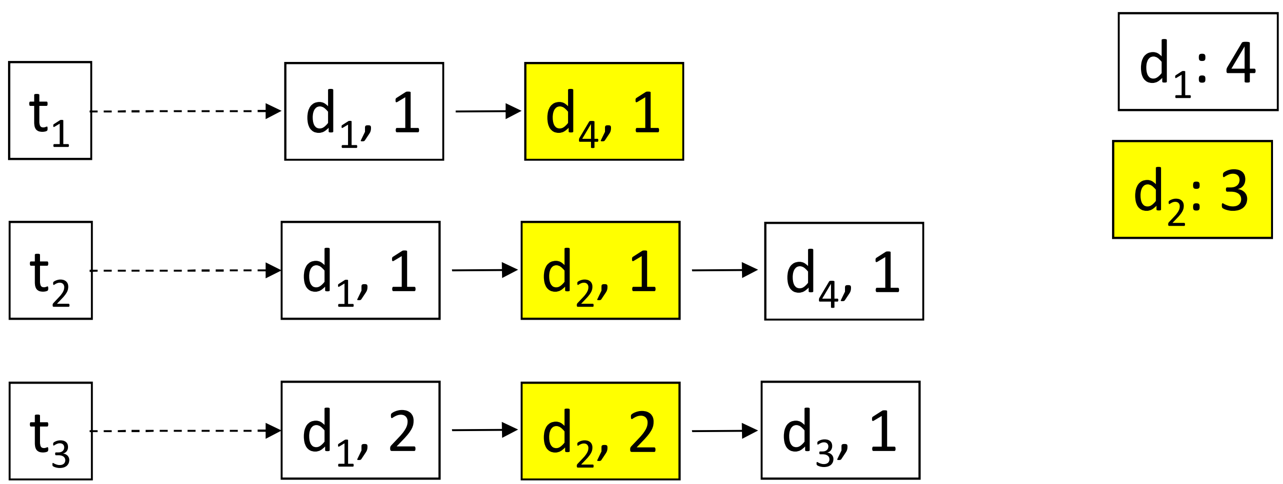
Document-at-a-time Ranking

- Read posting lists for query terms (t_1, …, $t_{|q|}$) concurrently
- Computes score when same document is seen in one or more posting lists
- Always advances posting list with lowest current document id
- Top-k results can be determined by keeping results in priority queue



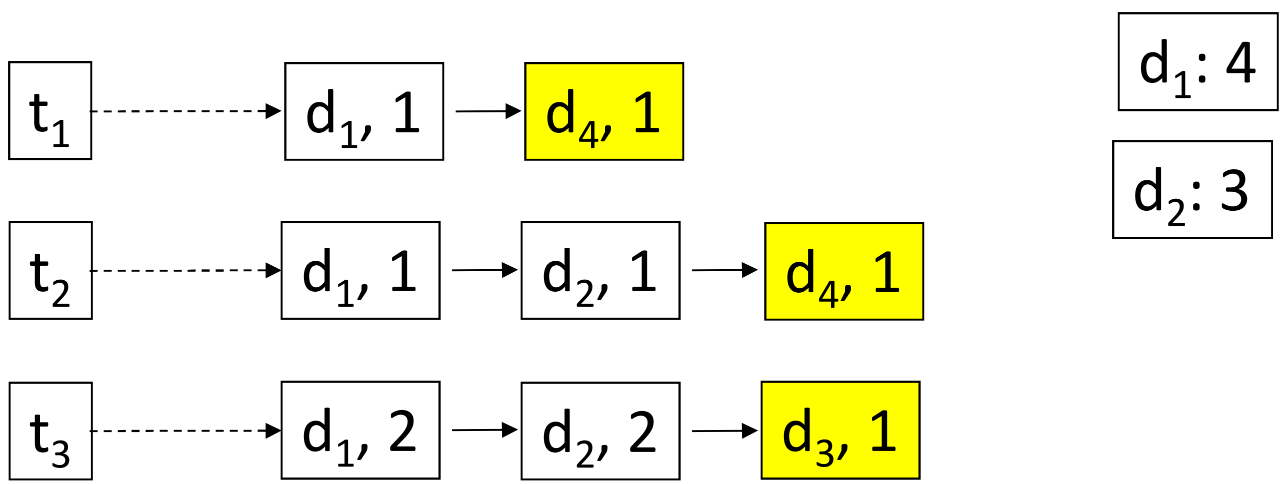
Document-at-a-time Ranking

- Read posting lists for query terms (t_1, …, $t_{|q|}$) concurrently
- Computes score when same document is seen in one or more posting lists
- Always advances posting list with lowest current document id
- Top-k results can be determined by keeping results in priority queue



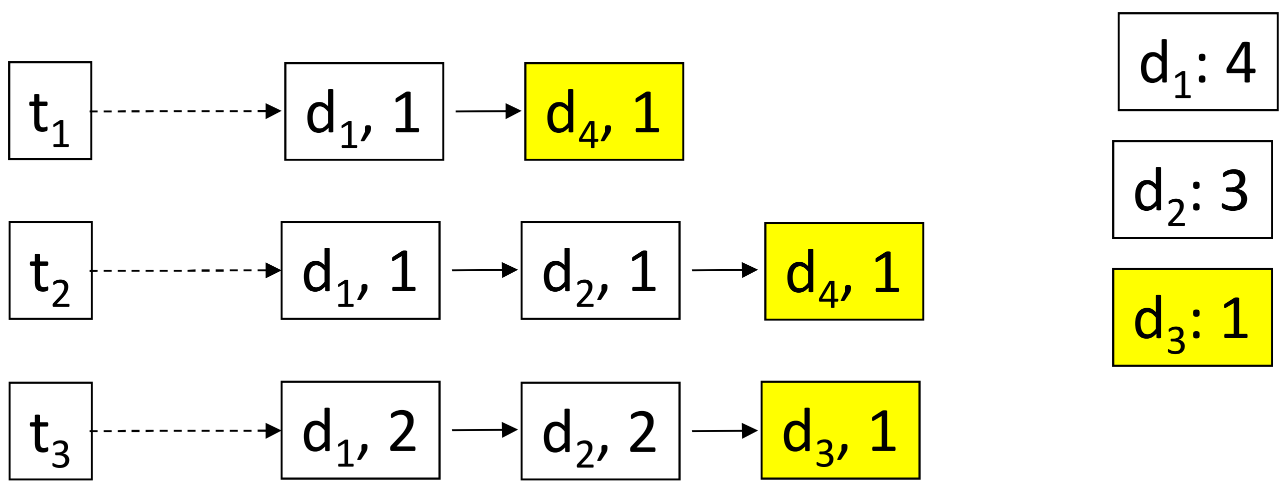
Document-at-a-time Ranking

- Read posting lists for query terms (t_1, …, $t_{|q|}$) concurrently
- Computes score when same document is seen in one or more posting lists
- Always advances posting list with lowest current document id
- Top-k results can be determined by keeping results in priority queue



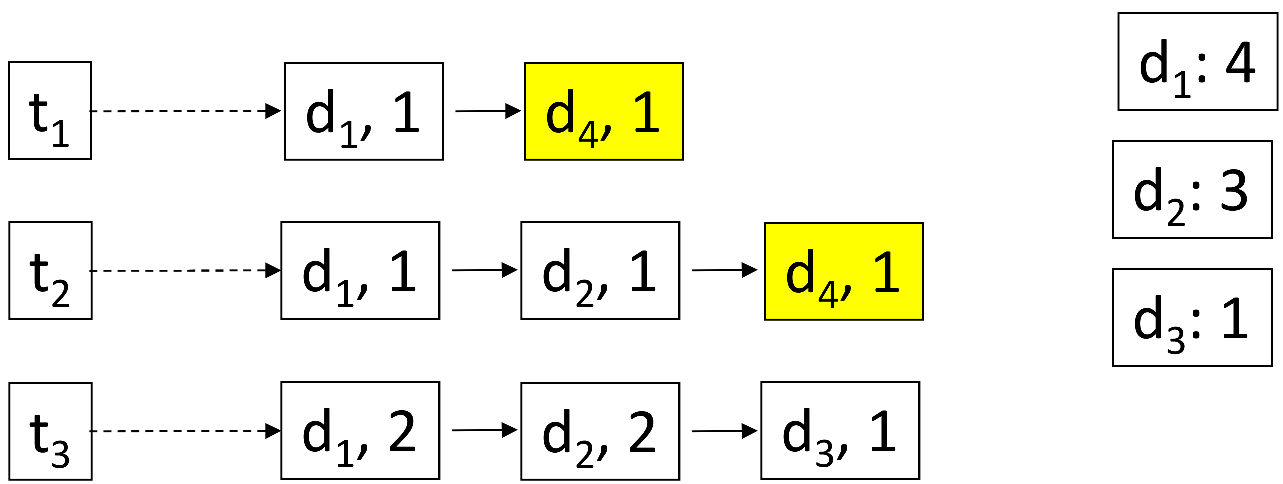
Document-at-a-time Ranking

- Read posting lists for query terms (t_1, …, $t_{|q|}$) concurrently
- Computes score when same document is seen in one or more posting lists
- Always advances posting list with lowest current document id
- Top-k results can be determined by keeping results in priority queue



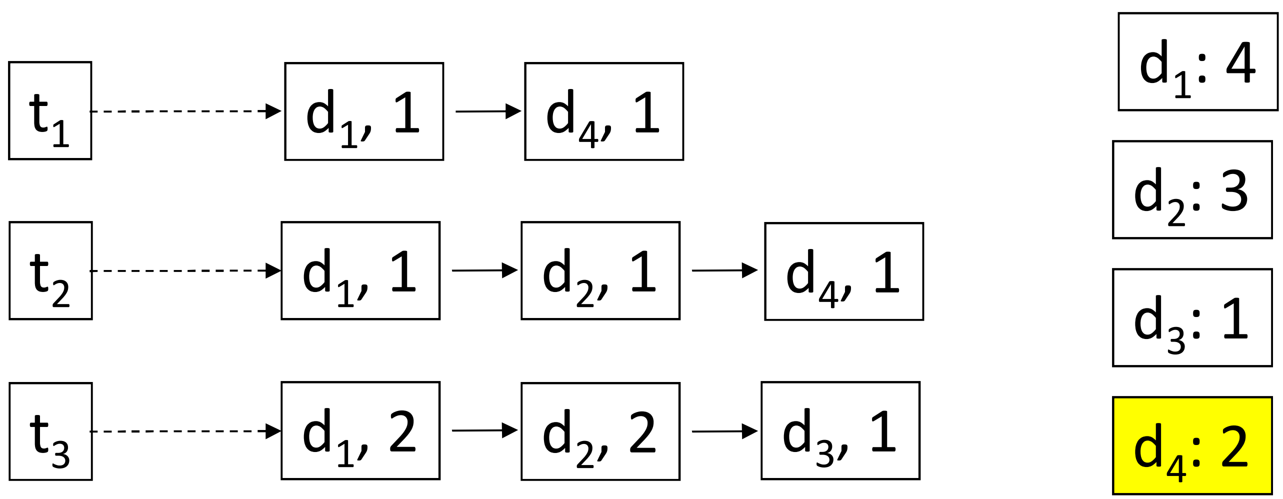
Document-at-a-time Ranking

- Read posting lists for query terms (t_1, …, $t_{|q|}$) concurrently
- Computes score when same document is seen in one or more posting lists
- Always advances posting list with lowest current document id
- Top-k results can be determined by keeping results in priority queue



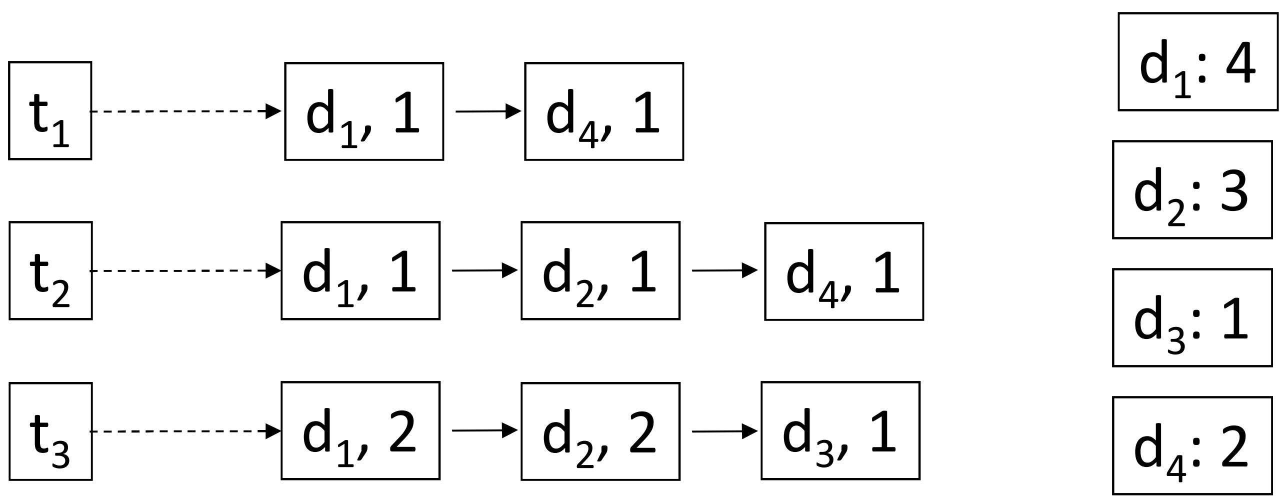
Document-at-a-time Ranking

- Read posting lists for query terms (t_1, …, $t_{|q|}$) concurrently
- Computes score when same document is seen in one or more posting lists
- Always advances posting list with lowest current document id
- Top-k results can be determined by keeping results in priority queue



Document-at-a-time Ranking

- Read posting lists for query terms (t_1, …, $t_{|q|}$) concurrently
- Computes score when same document is seen in one or more posting lists
- Always advances posting list with lowest current document id
- Top-k results can be determined by keeping results in priority queue



Document-at-a-time Ranking

```
context = {}        // maps a document to a list of matching terms
for w ∈ q do
    for d, count ∈ Idx.fetch_docs(w) do
        context[d].append(count)
    end for
end for
priority_queue = {}        // low score is treated as high priority
for d, term_counts ∈ context do
    score = 0
    for count ∈ term_counts do
        score = score + score_term(count)
    end for
    priority_queue.push(d, score)
    if priority_queue.size() > k then
        priority_queue.pop()        // removes lowest score so far
    end if
end for
Return sorted documents from priority_queue
```

References

- ChengXiang Zhai and Sean Massung, *Text Data Management and Analysis: A Practical Introduction to Information Retrieval and Text Mining*, ACM Books, 2016.
  - Chapter 6, Section 6.1-6.3 (Vector space model)
  - Chapter 8, Section 8.1-8.3 (Vector space model implementation)

Questions