WAND Top-K Retrieval

Alfan F. Wicaksono

Fakultas Ilmu Komputer, Universitas Indonesia

Berdasarkan paper "Efficient Query Evaluation using a Two-Level Retrieval Process", by Andrei Z. Broder, D. Carmel, M. Herscovici, A. Soffer, J. Zien

Ada 22 Milyar hasil yang relevan, namun yang ditampilkan hanya sekitar Top-10 saja.

Google

to be or not to be

Login

Semua Gambar Video Buku Berita Lainnya Alat SafeSearch aktif

Sekitar 22.810.000.000 hasil (0,55 detik)

Kiat: Telusuri hasil dalam bahasa **Indonesia** saja. Anda dapat menentukan bahasa penelusuran di Preferensi

"To be, or not to be" is **the opening phrase of a soliloquy given by Prince Hamlet in the so-called "nunnery scene" of William Shakespeare's play Hamlet, Act 3, Scene 1**. In the speech, Hamlet contemplates death and suicide, bemoaning the pain and unfairness of life but acknowledging that the alternative might be worse.

https://en.wikipedia.org › wiki › To_be,_or_not_to_be

To be, or not to be - Wikipedia

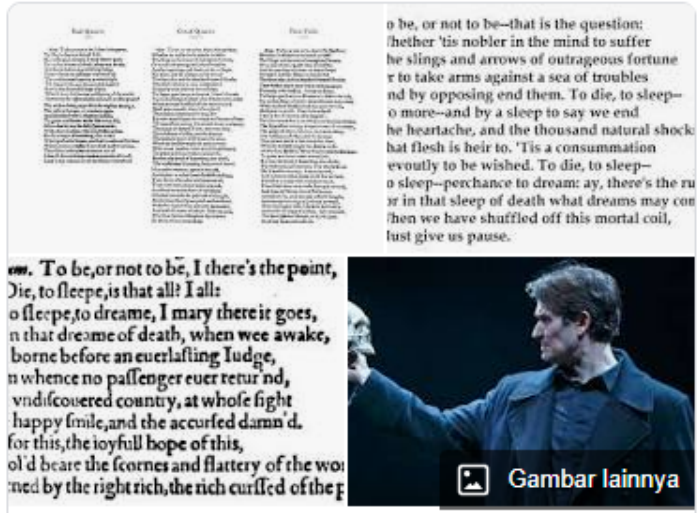
Tentang cuplikan pilihan • Masukan

Orang juga bertanya

What is the full quote To be, or not to be?

Why does Hamlet say To be, or not to be?

What does Shakespeare say To be, or not to be?



To be, or not to be

Diterjemahkan dari bahasa Inggris -
"Menjadi, atau tidak menjadi" adalah frasa pembuka dari solilokui yang diberikan oleh Pangeran Hamlet dalam apa yang disebut "adegan biara" dari drama William Shakespeare Hamlet, Act 3, Scene 1.

Wow! Hanya 0.55 detik !

Google

to be or not to be

Login

Semua Gambar Video Buku Berita Lainnya Alat SafeSearch aktif

Sekitar 22.810.000.000 hasil (0,55 detik)

Kiat: Telusuri hasil dalam bahasa **Indonesia** saja. Anda dapat menentukan bahasa penelusuran di Preferensi

"To be, or not to be" is **the opening phrase of a soliloquy given by Prince Hamlet in the so-called "nunnery scene" of William Shakespeare's play Hamlet, Act 3, Scene 1**. In the speech, Hamlet contemplates death and suicide, bemoaning the pain and unfairness of life but acknowledging that the alternative might be worse.

https://en.wikipedia.org › wiki › To_be,_or_not_to_be

To be, or not to be - Wikipedia

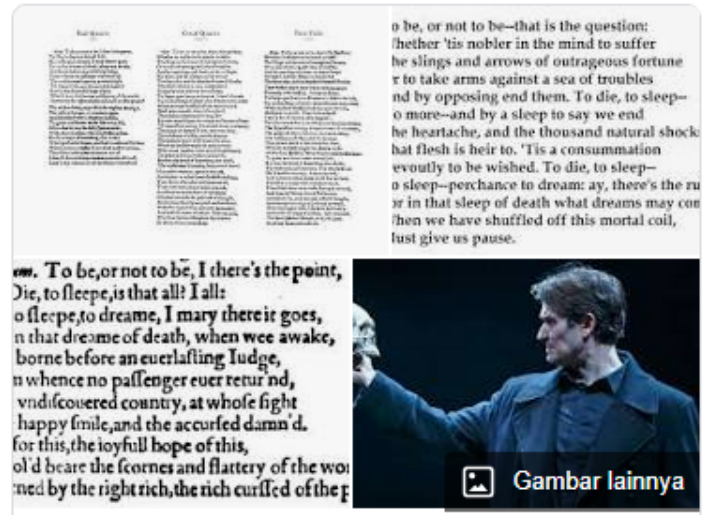
Tentang cuplikan pilihan • Masukan

Orang juga bertanya

What is the full quote To be, or not to be?

Why does Hamlet say To be, or not to be?

What does Shakespeare say To be, or not to be?



To be, or not to be

Diterjemahkan dari bahasa Inggris -
"Menjadi, atau tidak menjadi" adalah frasa pembuka dari solilokui yang diberikan oleh Pangeran Hamlet dalam apa yang disebut "adegan biara" dari drama William Shakespeare Hamlet, Act 3, Scene 1.

Scoring Method

$$score(Q, D) = \sum_{t \in Q \cap D} \alpha_t \, w(t, D)$$

Bagian yang tidak bergantung pada dokumen (bergantung pada query)

Bagian yang bergantung pada dokumen D

Scoring Method

$$score(Q, D) = \sum_{t \in Q \cap D} \alpha_t \, w(t, D)$$

Contoh: **TF-IDF**

$\alpha_t = tf(t, Q).log(N/df(t))$; boleh juga dengan $\alpha_t = log(N/df(t))$

$w(t, D) = tf(t, D)$; boleh juga dengan $w(t, D) = 1 + \log(tf(t, D))$

Juga dengan varian TF-IDF yang lainnya

Scoring Method

$$score(Q, D) = \sum_{t \in Q \cap D} \alpha_t \, w(t, D)$$

Contoh: **BM25**

$$\alpha_t = log(N/df(t))$$

$$w(t, D) = \frac{(k_1 + 1).tf(t, D)}{k_1 \left((1 - b) + b \frac{dl}{avdl} \right) + tf(t, D)}$$

Juga dengan varian BM25 yang lainnya

Term-at-a-Time (TaaT)

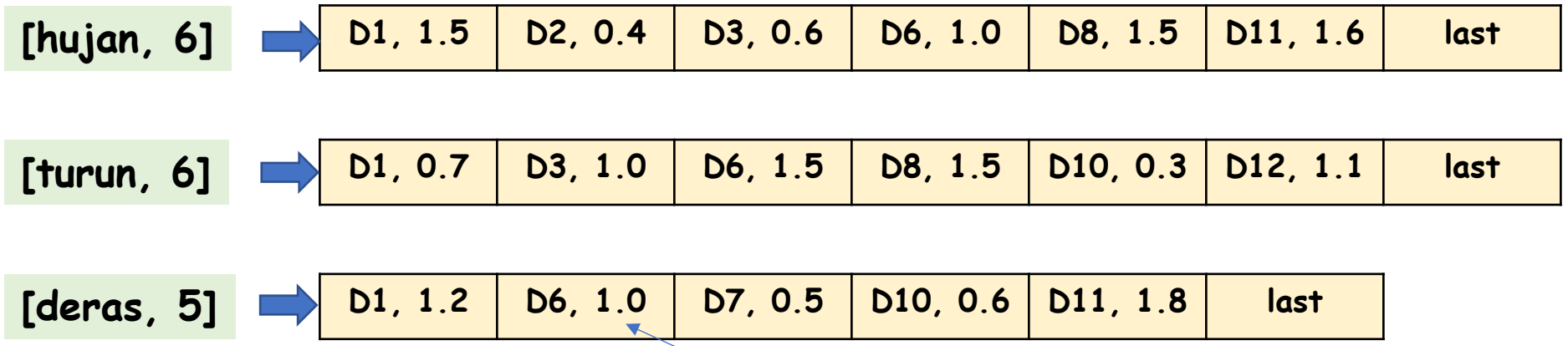
```
SCORE(q):
1   float Scores[N] = 0
2
3   for each query term t do
4       calculate w(t,q) and fetch postings list for t
5       for each pair(d, tf(t,d)) in postings list do
6           Scores[d] += w(t,d) × w(t,q)
7
8   return Top K components of Scores[]
```

Query: hujan turun deras

Term at a time

yang di simpen bukan score w(t, d) tapi gak disimpen di inverted index karena bnetuknya floating point number.

term, frequency

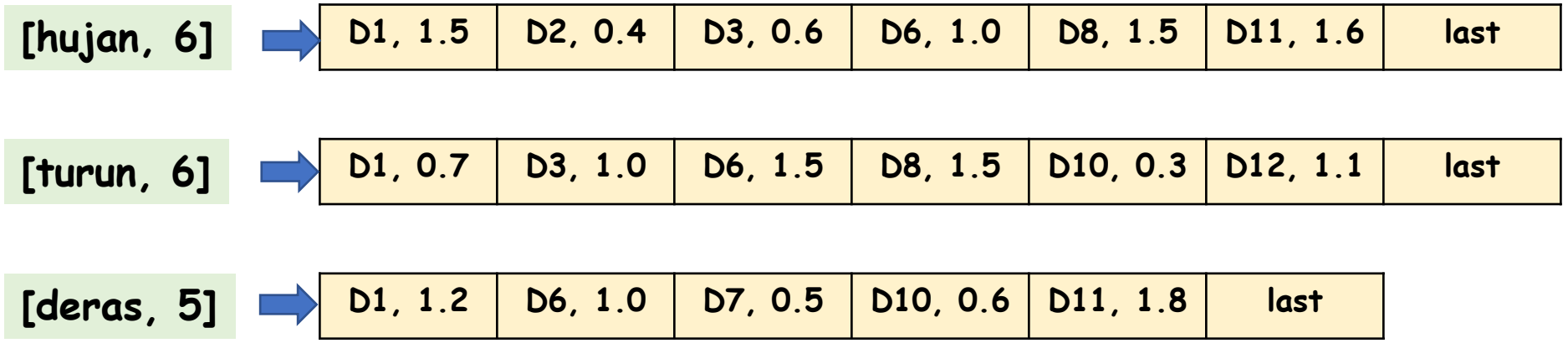


Misal, score **w(t,d)** sudah dihitung (bisa dengan TF-IDF, BM25 atau yang lainnya). Ini hanya untuk kemudahan saat visualisasi. Kenyataannya, nilai ini bukan sesuatu yang pre-computed dan disimpan di postings lists.

Yang pre-computed dan simpan biasanya adalah **tf(t,d)** yang berupa integer.

Query: hujan turun deras

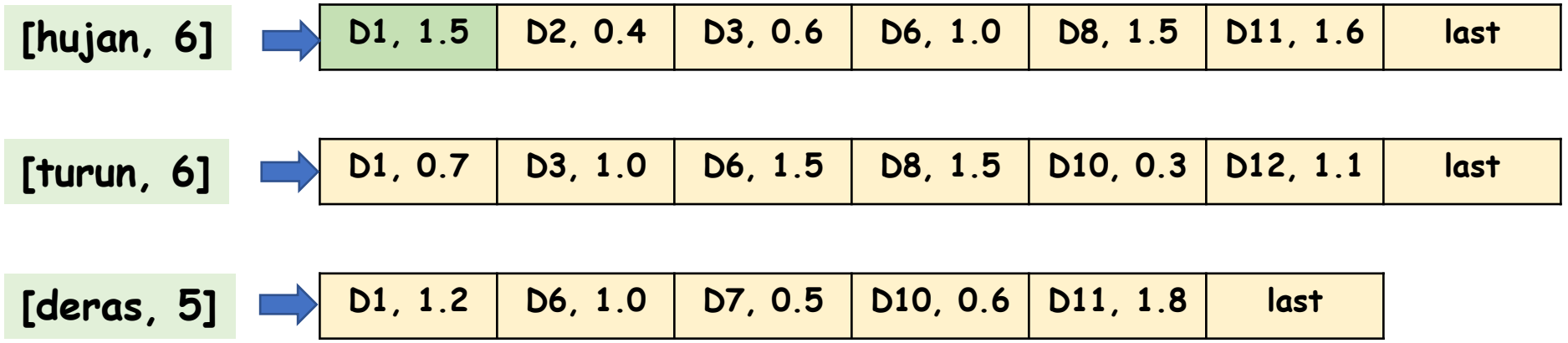
TaaT



Scores = []

Query: hujan turun deras

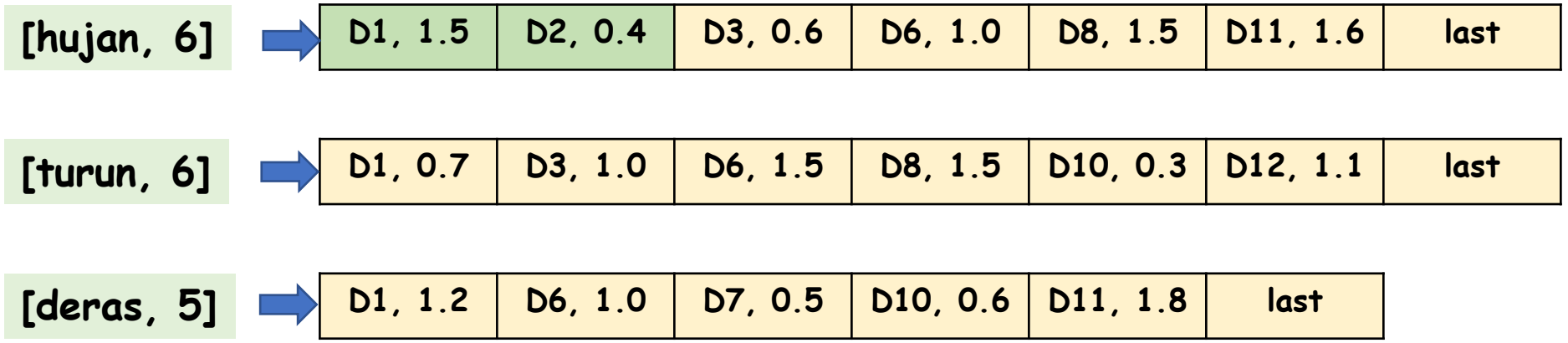
TaaT



Scores = [(D1, 1.5)]

Query: hujan turun deras

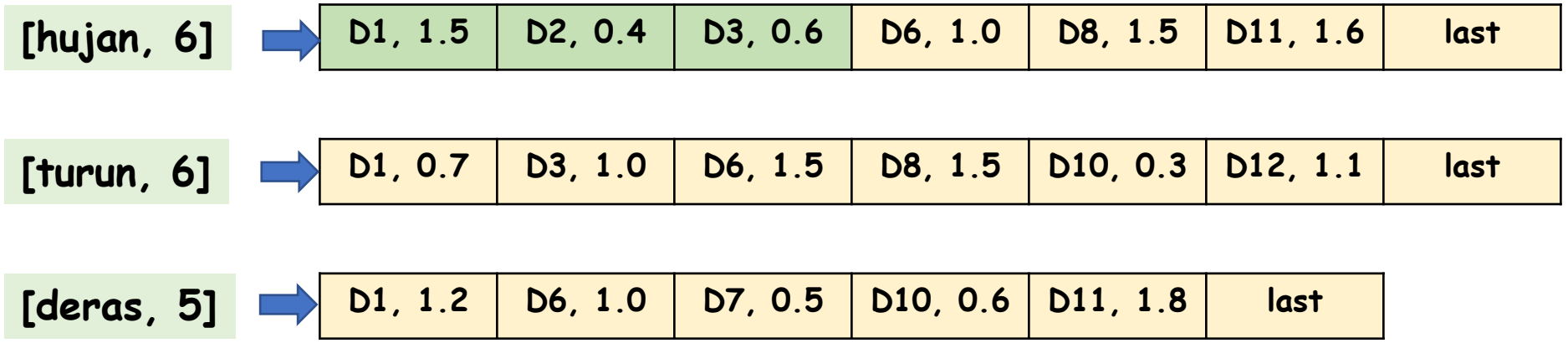
TaaT



Scores = [(D1, 1.5), (D2, 0.4),]

Query: hujan turun deras

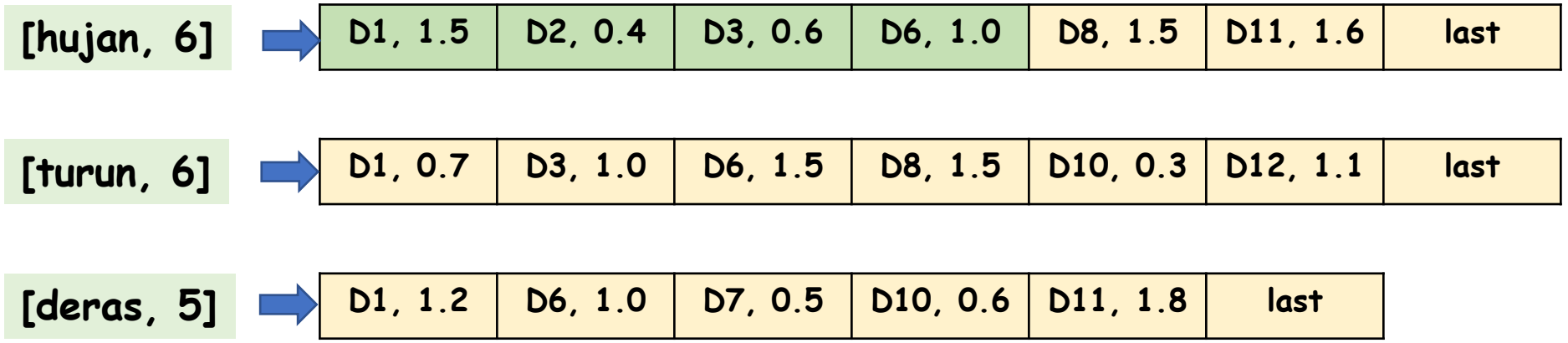
TaaT



Scores = [(D1, 1.5), (D2, 0.4), (D3, 0.6)]

Query: hujan turun deras

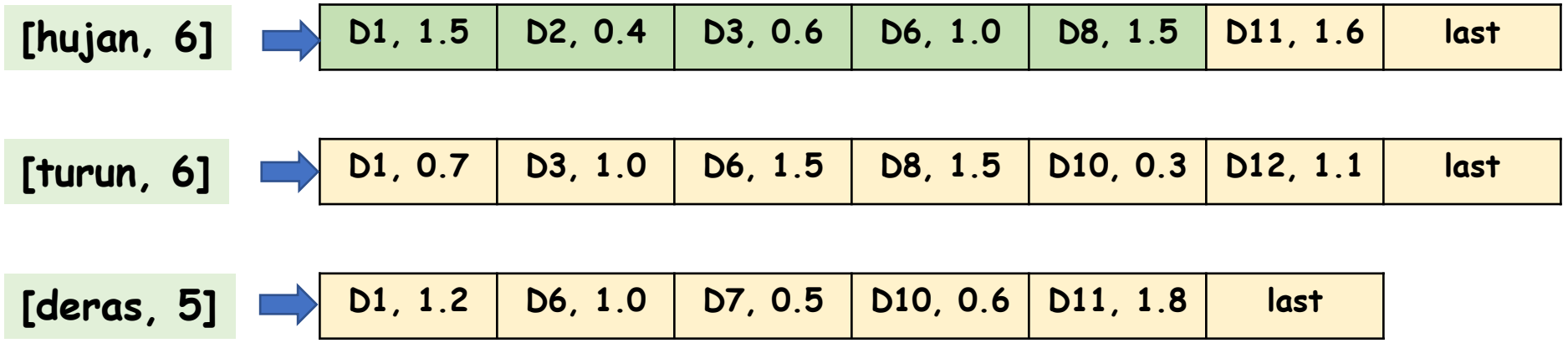
TaaT



Scores = [(D1, 1.5), (D2, 0.4), (D3, 0.6), (D6, 1.0)]

Query: hujan turun deras

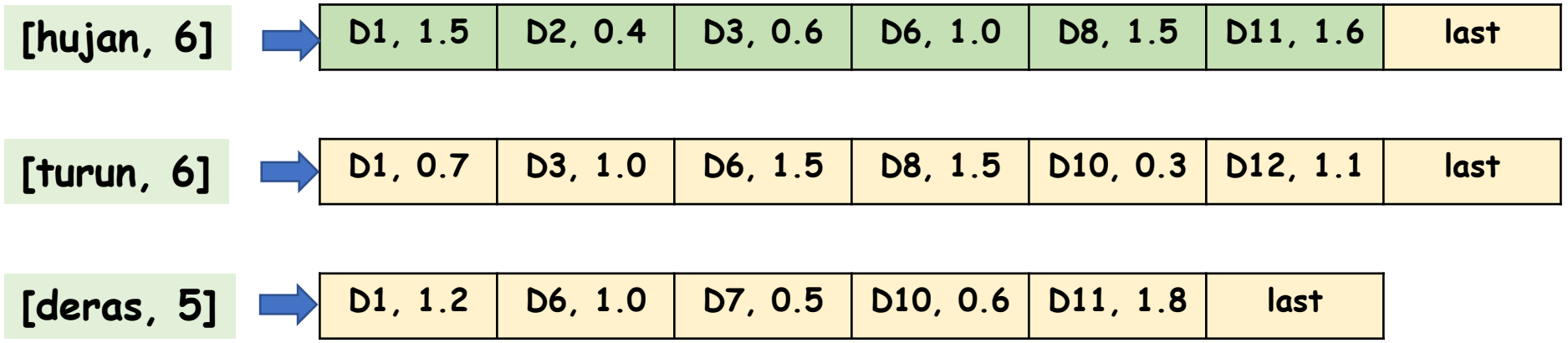
TaaT



Scores = [(D1, 1.0), (D2, 0.4), (D3, 0.6), (D6, 1.0), (D8, 1.5)]

Query: hujan turun deras

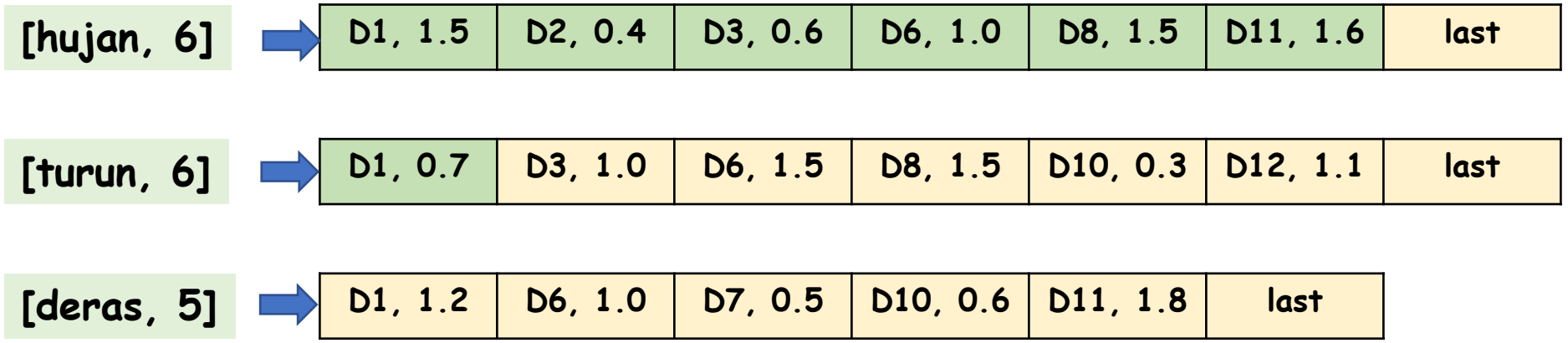
TaaT



Scores = [(D1, 1.5), (D2, 0.4), (D3, 0.6), (D6, 1.0), (D8, 1.5), (D11, 1.6)]

Query: hujan turun deras

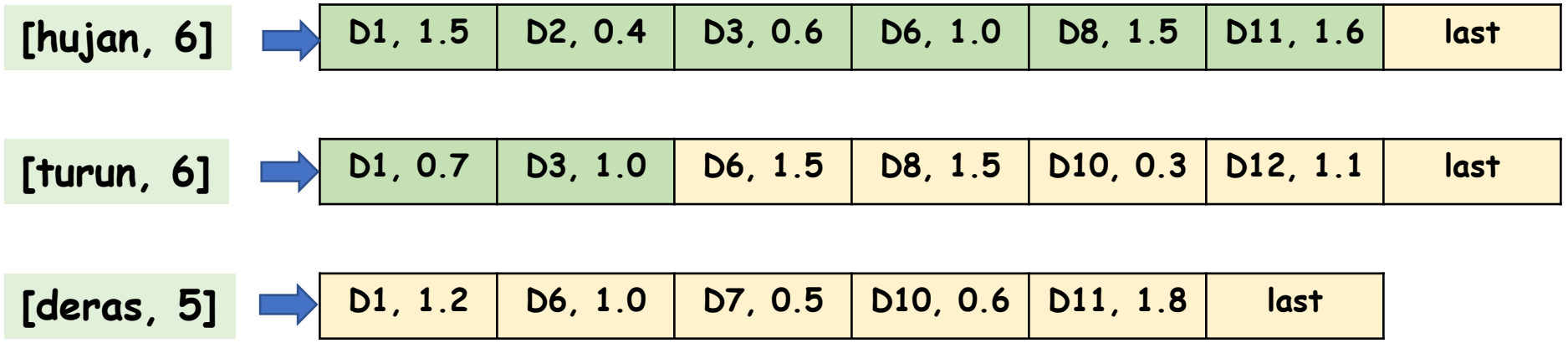
TaaT



Scores = [(D1, 2.2), (D2, 0.4), (D3, 0.6), (D6, 1.0), (D8, 1.5), (D11, 1.6)]

Query: hujan turun deras

TaaT



Scores = [(D1, 2.2), (D2, 0.4), (D3, 1.6), (D6, 1.0), (D8, 1.5), (D11, 1.6)]

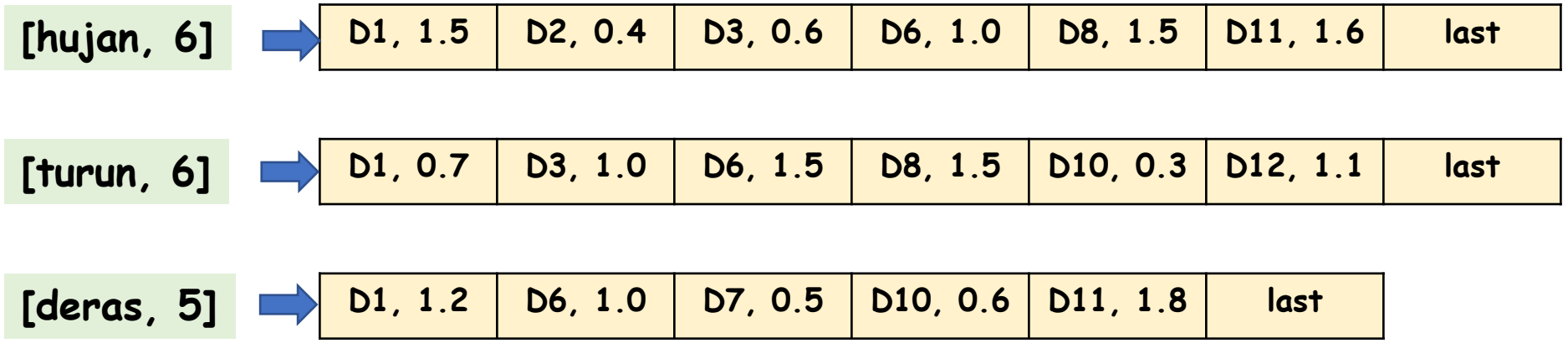
Dan, seterusnya, silakan lanjutkan sendiri ...
Jika score semua dokumen sudah dihitung, **gunakan HEAP untuk cari Top-K**

Document-at-a-Time (DaaT)

- Semua postings lists yang terkait query diproses secara paralel.
- Score dari sebuah dokumen dihitung secara penuh (fully evaluated) sebelum berpindah ke dokumen yang lain.
- Di setiap list ada pointer ke current Doc ID. Selalu majukan pointer pada list dengan currect Doc ID paling kecil.

Query: hujan turun deras

DaaT

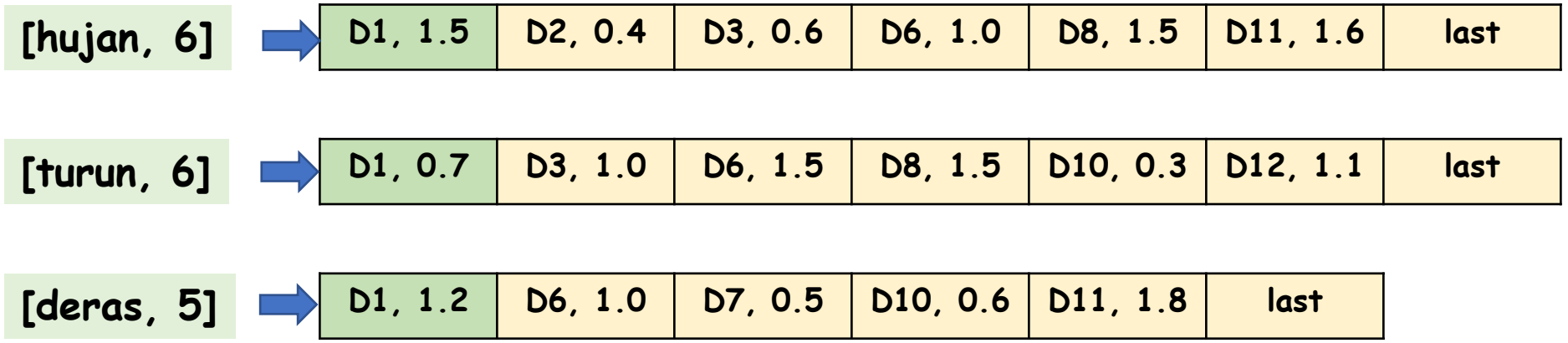


Scores = []

Query: hujan turun deras

DaaT

Jumlah dokumen yang fully-evaluated: 1



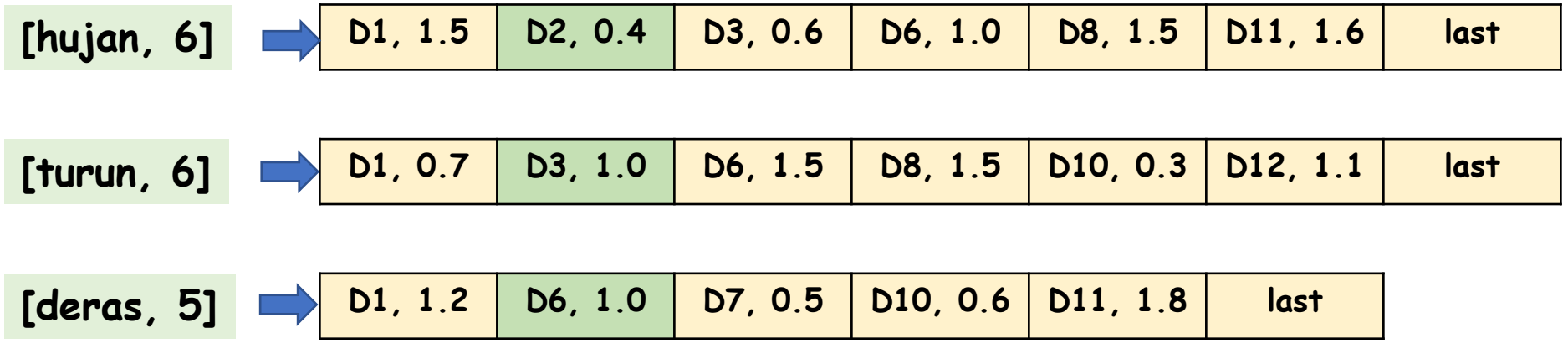
3.4 itu dapet dari 1.5 + 0.7 + 1.2 (hal ini karena D1 semuanya)

Scores = [(D1, 3.4),]

Query: hujan turun deras

DaaT

Jumlah dokumen yang fully-evaluated: 2



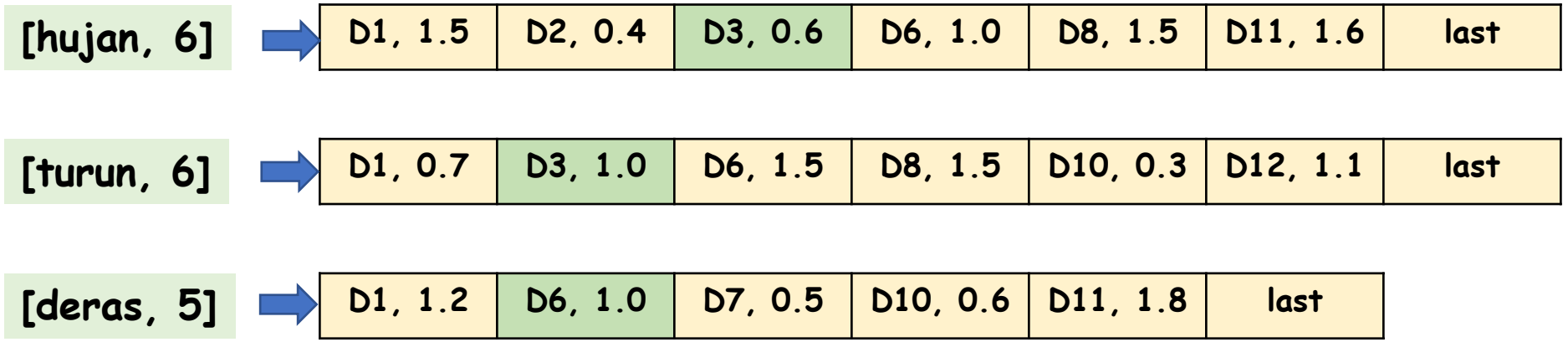
Kalau misal ada D2, D3, D6, kita pilih yang paling kecil index dokumennya nya. Nah baru kita masukkin (pointer yang D2 dimajuin)

Scores = [(D1, 3.4), (D2, 0.4)]

Query: hujan turun deras

DaaT

Jumlah dokumen yang fully-evaluated: 3



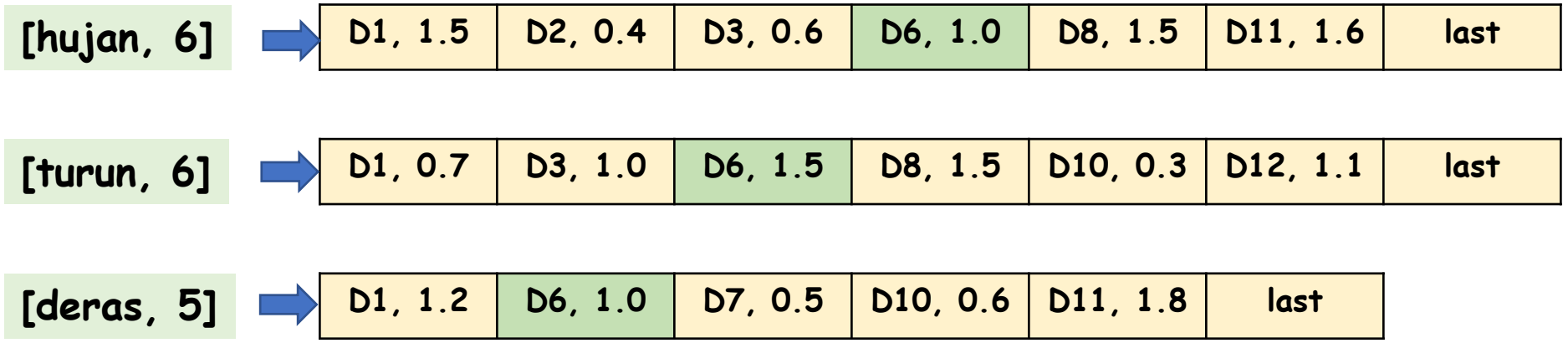
Sekarang yang paling kecil itu D3, nah kita ambil 0.6 + 1, terus majuin lagi pointernya

Scores = [(D1, 3.4), (D2, 0.4), (D3, 1.6)]

Query: hujan turun deras

DaaT

Jumlah dokumen yang fully-evaluated: 4



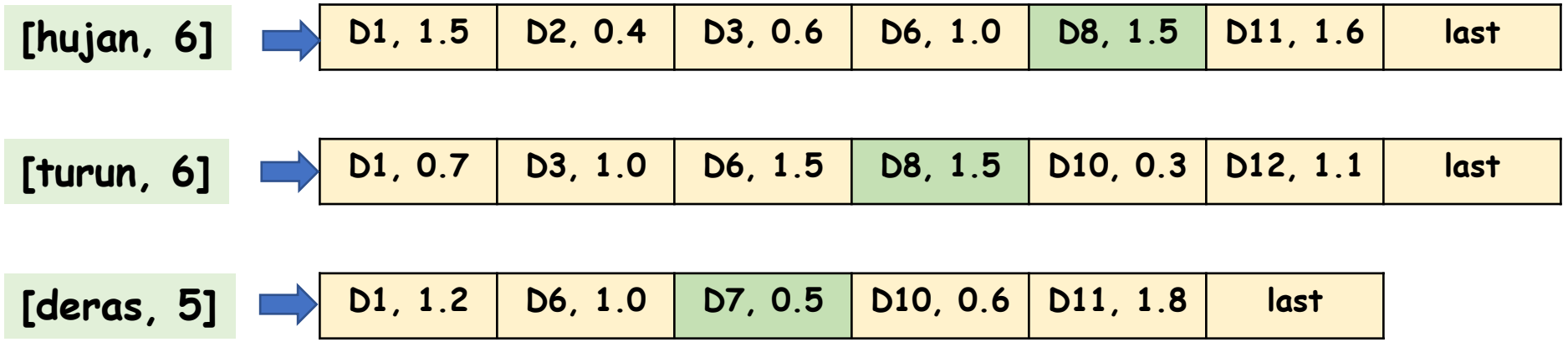
3 3 nya pointenrya majuin

Scores = [(D1, 3.4), (D2, 0.4), (D3, 1.6), (D6, 3.5)]

Query: hujan turun deras

DaaT

Jumlah dokumen yang fully-evaluated: 5

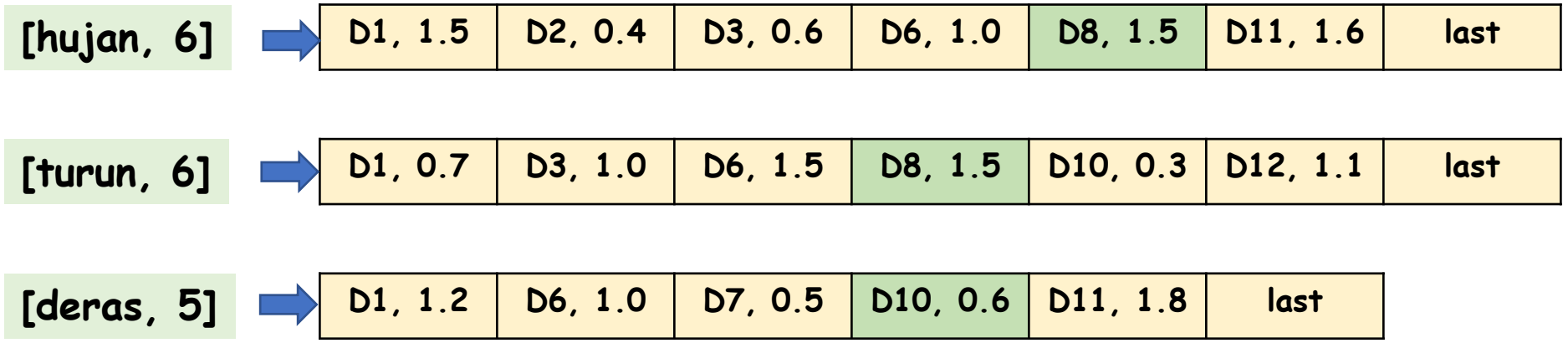


Scores = [(D1, 3.4), (D2, 0.4), (D3, 1.6), (D6, 3.5), (D7, 0.5)]

Query: hujan turun deras

DaaT

Jumlah dokumen yang fully-evaluated: 6

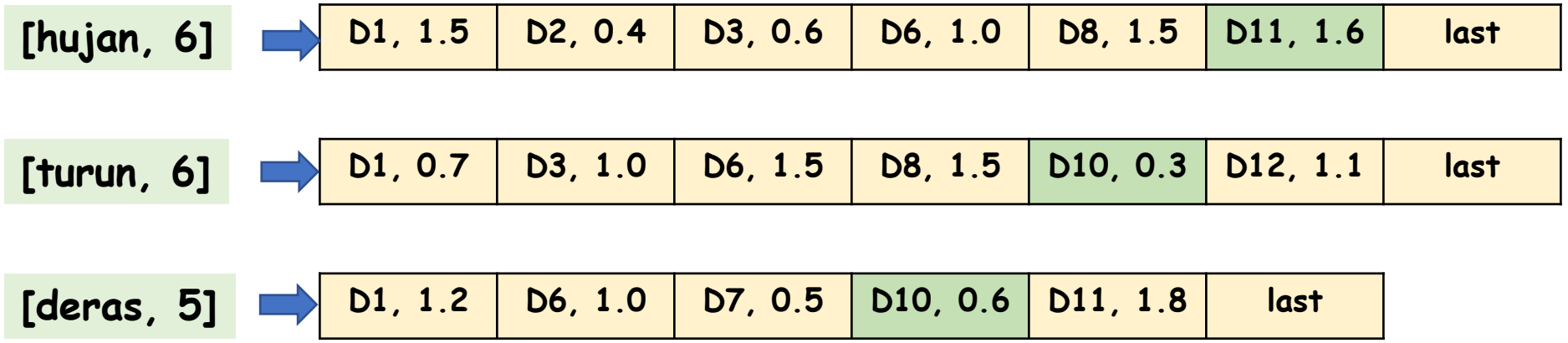


Scores = [(D1, 3.4), (D2, 0.4), (D3, 1.6), (D6, 3.5), (D7, 0.5), (D8, 3.0)]

Query: hujan turun deras

DaaT

Jumlah dokumen yang fully-evaluated: 7

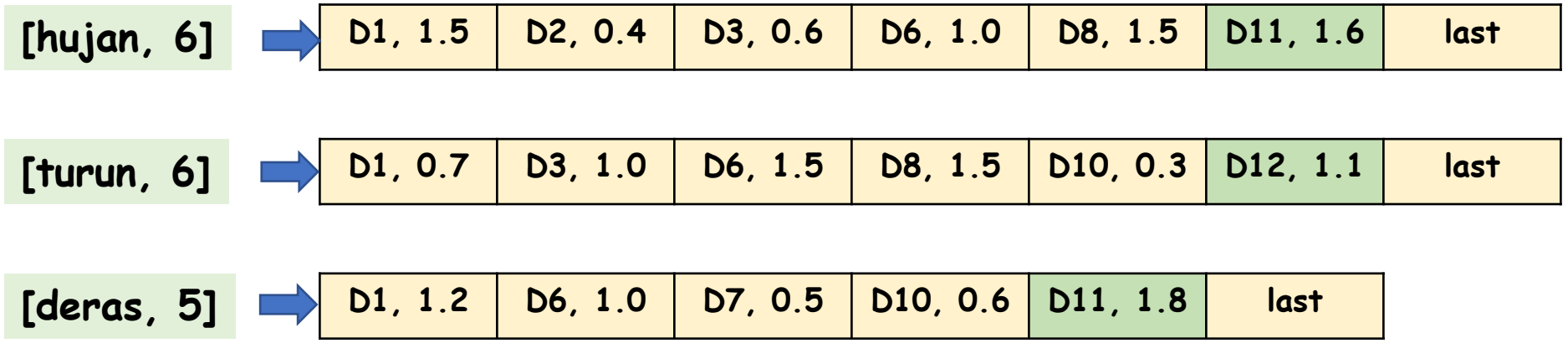


Scores = [(D1, 3.4), (D2, 0.4), (D3, 1.6), (D6, 3.5), (D7, 0.5), (D8, 3.0), (D10, 0.9)]

Query: hujan turun deras

DaaT

Jumlah dokumen yang fully-evaluated: 8

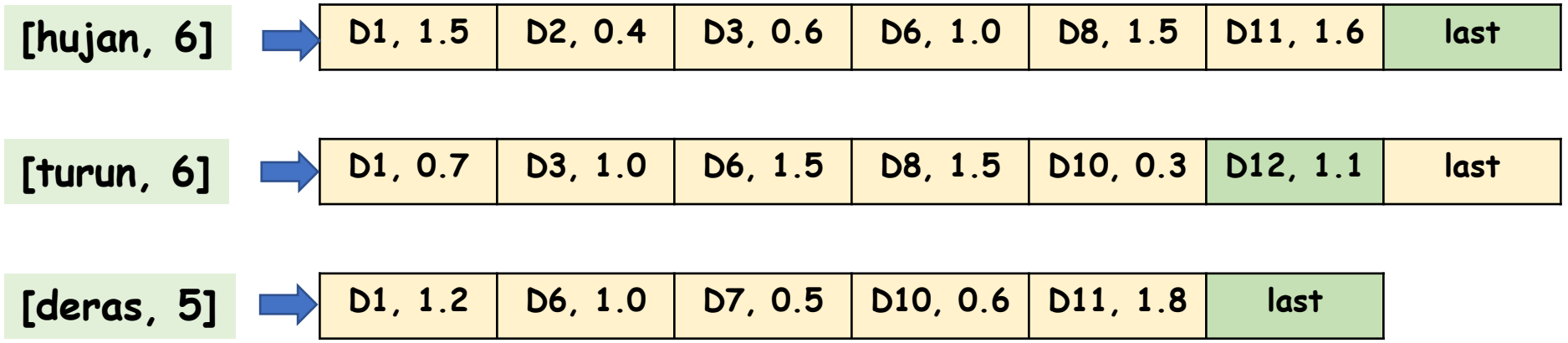


Scores = [(D1, 3.4), (D2, 0.4), (D3, 1.6), (D6, 3.5), (D7, 0.5), (D8, 3.0), (D10, 0.9), (D11, 3.4)]

Query: hujan turun deras

DaaT

Jumlah dokumen yang fully-evaluated: 9

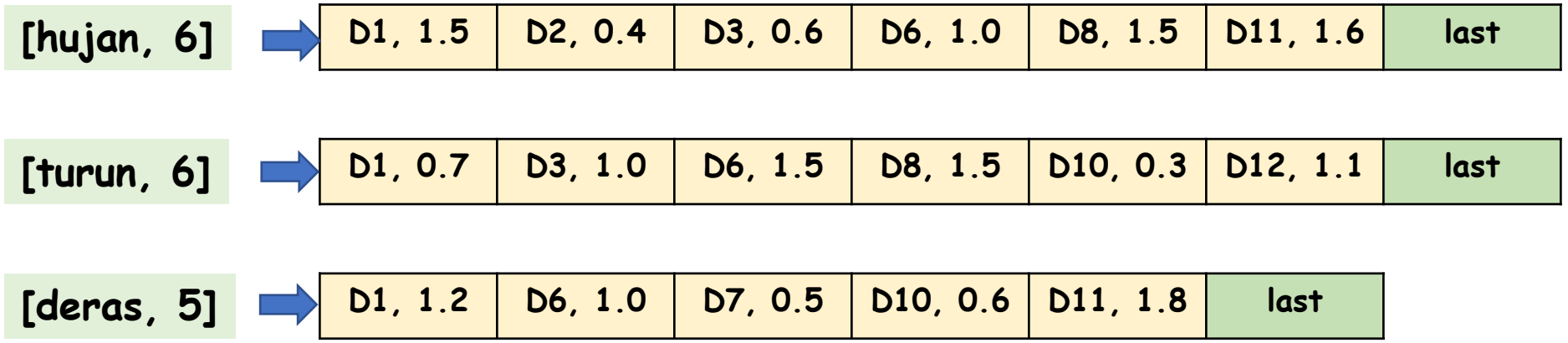


Scores = [(D1, 3.4), (D2, 0.4), (D3, 1.6), (D6, 3.5), (D7, 0.5), (D8, 3.0), (D10, 0.9), (D11, 3.4), (D12, 1.1)]

Query: hujan turun deras

DaaT

Jumlah dokumen yang fully-evaluated: 9



Scores = [(D1, 3.4), (D2, 0.4), (D3, 1.6), (D6, 3.5), (D7, 0.5), (D8, 3.0), (D10, 0.9), (D11, 3.4), (D12, 1.1)]

→ Top-K HEAP

Masalah dengan DaaT sebelumnya

Best casenya itu O(jumlah dokumen)

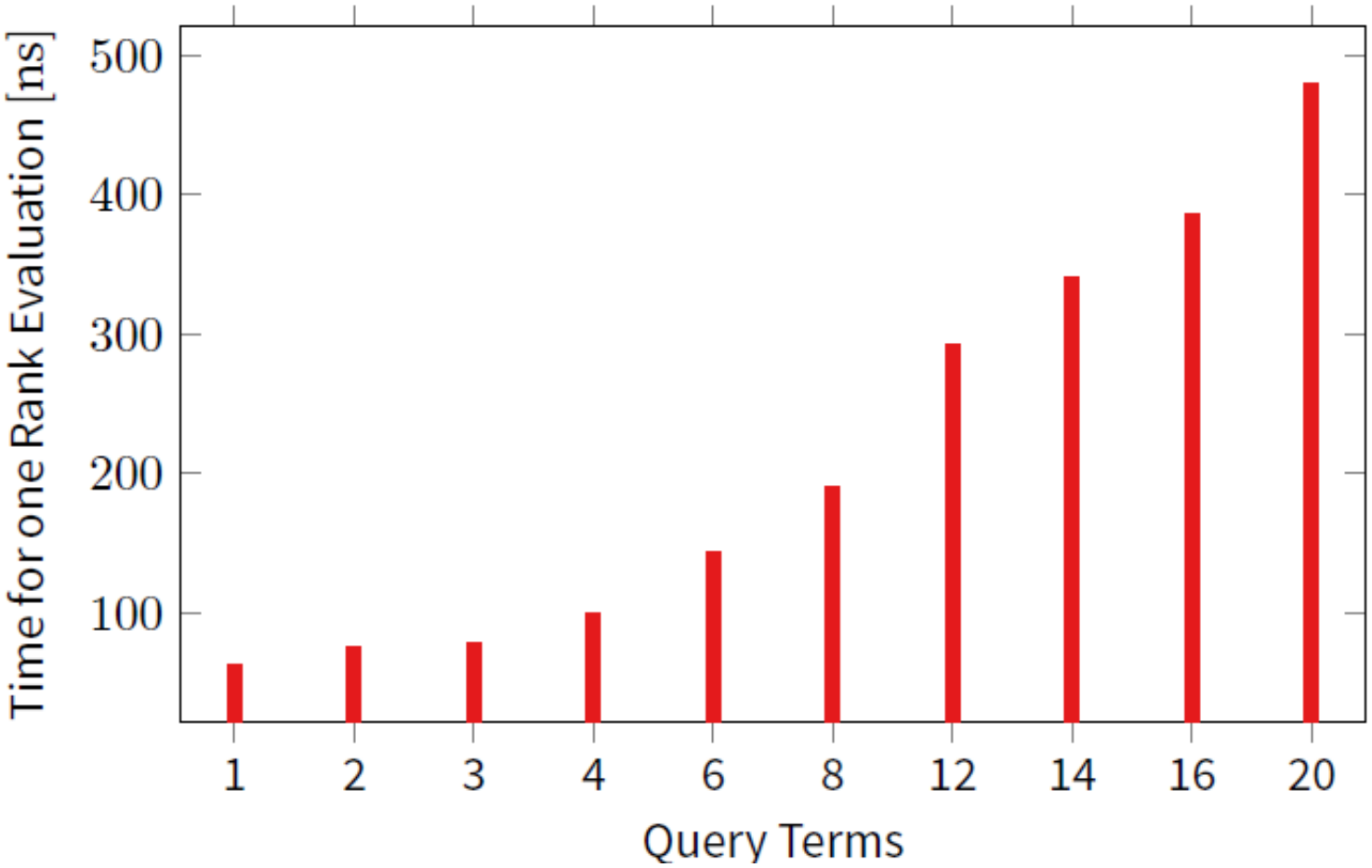
- Semua dokumen harus dihitung score-nya (fully-evaluated)

- Menghitung score dari sebuah dokumen (misal dengan BM25) sangat mahal! (lihat slide berikutnya)

Sangat mahal hitung score tapi kalo querynya term nya makin banyak maka makin mahal

- Bisakah kita mencari Top-K documents **tanpa harus menghitung score dari semua dokumen** pada postings lists terkait?

BM25 Computation Speed



BM25 Computation Speed

- Menghitung BM25 untuk sebuah pasangan query dan dokumen memerlukan kira-kira **100 nanoseconds**.

- Misal, query **"the student"** cocok dengan **25M dokumen**. Kita tahu bahwa "the" bisa muncul di hampir setiap dokumen.

25.000.000 x 100 nanoseconds = 2.5 seconds

- Apakah user mau menunggu **2.5 detik** untuk dapat jawaban?

E. Schurman and J. Brutlag. “Performance related changes and their user impact”. Velocity, 2009.
R. Kohavi, A. Deng, B. Frasca, T. Walker, Y. Xu, and N. Pohlmann. “Online controlled experiments at large scale”. In Proc. KDD, pages 1168–1176, 2013.

A/B Testing Experiment by Google

- Average revenue per user **turun 4%** ketika presentasi hasil search diperlambat.
- User sadar atau sensitive terhadap search system yang diperlambat minimal **50 ms**.

Bing Search Engine

- Mengurangi waktu proses sebesar **100 ms** meningkatkan annual revenue sebesar **0.6%**.
- Di tahun 2015, **1% kenaikan revenue** per user setara dengan kenaikan sekitar **$10.000.000** per tahun.

Strategi

Tambah server (horizontal scaling)

- Server bertambah -> perlu usaha maintenance lebih, system makin kompleks, power cost yang makin naik.

Ganti spesifikasi hardware yang lebih canggih (vertical scaling)

- Harga mahal & tidak scaling terus-menerus (diminishing returns)

Perbaikan masih bisa di level Software/Program

- Usulkan algoritma query processing yang lebih cepat atau efisien

Solusi: Top-K Retrieval

- Secara umum, search engine memang tidak menampilkan semua dokumen yang relevan; biasanya hanya Top-K saja (dengan K biasanya 10).

- For a given query, it's possible to produce a "complete" ranking without scoring all documents in the postings lists.

- Avoid scoring documents that we know will not appear in the Top-K results.

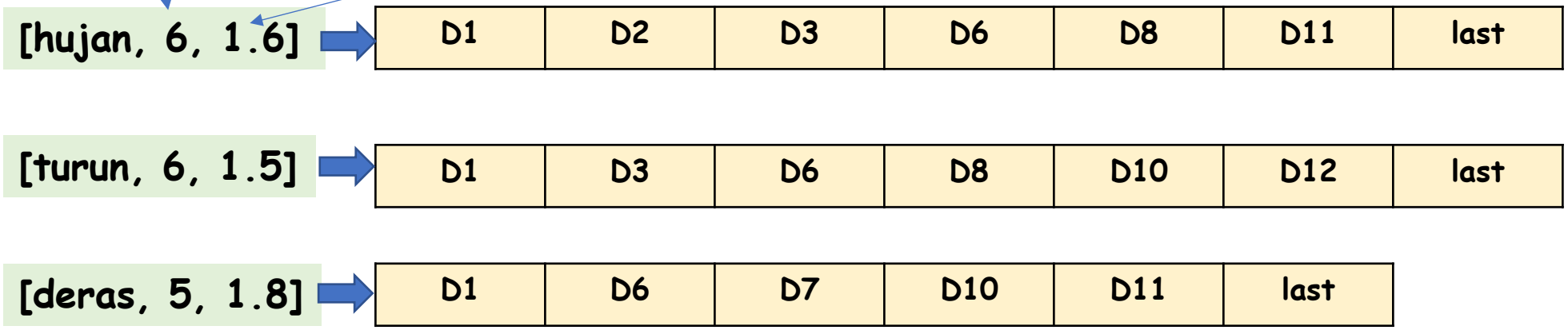
Query: hujan turun deras

Upper Bound Score per Term
Term's maximum contribution to any document score in its postings list

$$UB_t \geq \alpha_t \max(w(t, d_1), w(t, d_2), \dots)$$

DF

set upperbound, dimana cuman return dokumen yang diatas upperbound



WAND Operator

Digunakan untuk menentukan apakah sebuah dokumen punya kemungkinan untuk berada di Top-K.

$$WAND(x_1, UB_1, x_2, UB_2, x_3, UB_3, \dots, x_k, UB_k, \theta)$$

The minimum score in the current Top-K documents

= **True jika** $UB(d,q) = \sum_{1 \le i \le k} x_i UB_i \ge \theta$

x_i bernilai 1 jika term **i** ada di dokumen; dan 0 jika sebaliknya.

di skip kalau dokumen gak mungkin melawan upperboud

Jika **True**, artinya dokumen tersebut punya chance untuk berada di Top-K.
Jika **False**, sudah tidak ada chance. **Jangan dievaluasi! Skip saja!**

Two Level Approach

Tahap 1 (WAND Condition)

- Cari kumpulan kandidat dokumen dengan menghitung UB(d,q)
- Jika UB(d,q) >= minimum score, ke tahap 2
- Jika tidak, di-skip

tidak perlu hitung dokumen yang bm25 nya kurang dari minimum score

Tahap 2 (Full Evaluation)

- Hitung exact full score dari para kandidat yang lolos tahap 1
- Periksa apakah bisa mengalahkan minimum score

Tahap Akhir (Top-K Heap)

TOP-1 :

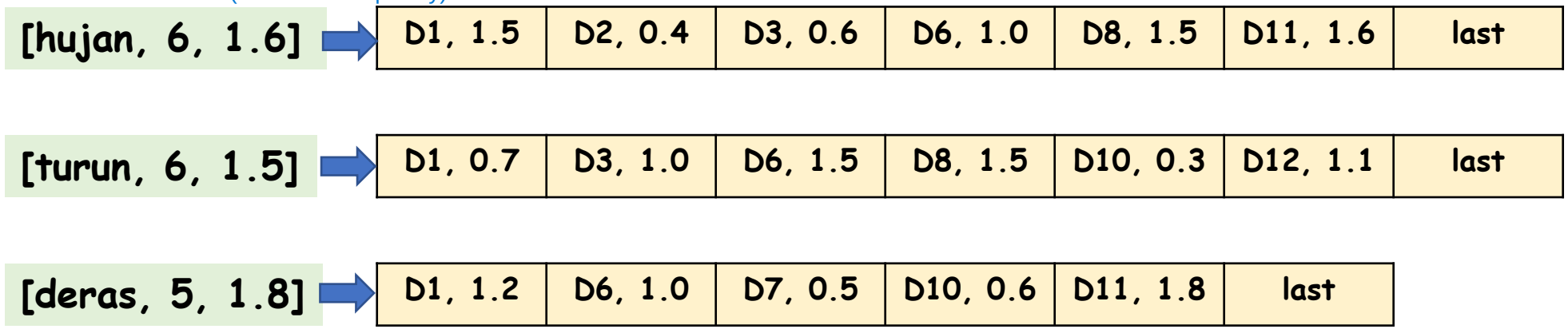
Threshold :

Pivot : posisi tengah (ada di sebelah kiri/sebelah kanan). Jadi ini buat nentuin posisi mana dokumen yang tidak layak/tidak layak masuk top K

curDoc : sebelum pivot tidak mungkin top K

urut
cari pivot
terus ada if yang dilalui, ikutin tahap bawah lagi

1.6 dari D11 itu upperbound, dihitung secara 1 off (jadi cuman sekali ktia evaluasi (yaiut pas training)
6 = DF (document frequency)



TOP-1 : []
Threshold : 0
Pivot : 0
curDoc : 0

untuk setiap ngitung dokumen pakai bm25, komputasinya lama

kita ingin untuk membuang dokumen yang pasti engga masuk ke top K

jadi kita cuman itung dokumen yang punya

kesempatan jadi top k, lalu dihitung bm25 nya

```
1. Function init(queryTerms)
2.     terms ← queryTerms
3.     curDoc ← 0
4.     for each t ∈ terms
5.         posting[t] ← t.iterator.next(0)
```

Inisialisasi

- Sets the current document (curDoc) to be considered to zero
- For each query term, t, initializes its current posting to be the first posting item in the list

[hujan, 6, 1.6] →	D1, 1.5	D2, 0.4	D3, 0.6	D6, 1.0	D8, 1.5	D11, 1.6	last
[turun, 6, 1.5] →	D1, 0.7	D3, 1.0	D6, 1.5	D8, 1.5	D10, 0.3	D12, 1.1	last
[deras, 5, 1.8] →	D1, 1.2	D6, 1.0	D7, 0.5	D10, 0.6	D11, 1.8	last	

```
1. Function next(θ)
2.    repeat
3.       /* Sort the terms in non decreasing order of
          DID */
4.       sort(terms, posting)
5.       /* Find pivot term - the first one with accumulated
          UB ≥ θ */
6.       pTerm ← findPivotTerm(terms, θ)
7.       if (pTerm = null) return (NoMoreDocs)
8.       pivot ← posting[pTerm].DID
9.       if (pivot = lastID) return (NoMoreDocs)
10.      if (pivot ≤ curDoc)
11.         /* pivot has already been considered, advance
             one of the preceding terms */
12.         aterm ← pickTerm(terms[0..pTerm])
13.         posting[aterm] ← aterm.iterator.next(curDoc+1)
14.      else /* pivot > curDoc */
15.         if (posting[0].DID = pivot)
16.            /* Success, all terms preceding pTerm belong
                to the pivot */
17.            curDoc ← pivot
18.            return (curDoc, posting)
19.         else
20.            /* not enough mass yet on pivot, advance
                one of the preceding terms */
21.            aterm ← pickTerm(terms[0..pTerm])
22.            posting[aterm] ← aterm.iterator.next(pivot)
23.      end repeat
```

TOP-1 : []
Threshold : 0
Pivot : 0
curDoc : 0

#documents that are fully evaluated: 0

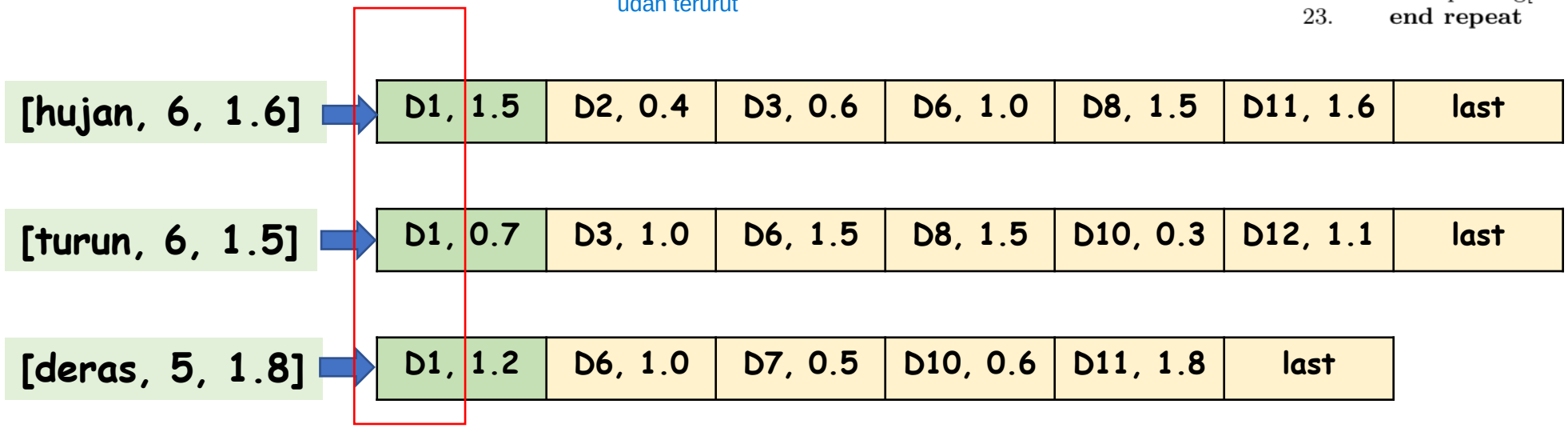
```
1. Function next(θ)
2.    repeat
3.       /* Sort the terms in non decreasing order of
          DID */
4.       sort(terms, posting)
5.       /* Find pivot term - the first one with accumulated
          UB ≥ θ */
6.       pTerm ← findPivotTerm(terms, θ)
7.       if (pTerm = null) return (NoMoreDocs)
8.       pivot ← posting[pTerm].DID
9.       if (pivot = lastID) return (NoMoreDocs)
10.      if (pivot ≤ curDoc)
11.         /* pivot has already been considered, advance
             one of the preceding terms */
12.         aterm ← pickTerm(terms[0..pTerm])
13.         posting[aterm] ← aterm.iterator.next(curDoc+1)
14.      else /* pivot > curDoc */
15.         if (posting[0].DID = pivot)
16.            /* Success, all terms preceding pTerm belong
                to the pivot */
17.            curDoc ← pivot
18.            return (curDoc, posting)
19.         else
20.            /* not enough mass yet on pivot, advance
                one of the preceding terms */
21.            aterm ← pickTerm(terms[0..pTerm])
22.            posting[aterm] ← aterm.iterator.next(pivot)
23.      end repeat
```

pivot term itu selalu lihat dari query
pivot itu pointer yang ada di pivot term

threashold di update dari maximum dokumen

Sudah terurut berdasarkan Doc ID

udah terurut



TOP-1 : []
Threshold : 0
Pivot : 0
curDoc : 0

#documents that are fully evaluated: 0

```
1. Function next(θ)
2.   repeat
3.     /* Sort the terms in non decreasing order of
          DID */
4.     sort(terms, posting)
5.     /* Find pivot term - the first one with accumulated
          UB ≥ θ */
6.     pTerm ← findPivotTerm(terms, θ)
7.     if (pTerm = null) return (NoMoreDocs)
8.     pivot ← posting[pTerm].DID
9.     if (pivot = lastID) return (NoMoreDocs)
10.    if (pivot ≤ curDoc)
11.      /* pivot has already been considered, advance
            one of the preceding terms */
12.      aterm ← pickTerm(terms[0..pTerm])
13.      posting[aterm] ← aterm.iterator.next(curDoc+1)
14.    else /* pivot > curDoc */
15.      if (posting[0].DID = pivot)
16.        /* Success, all terms preceding pTerm belong
              to the pivot */
17.        curDoc ← pivot
18.        return (curDoc, posting)
19.      else
20.        /* not enough mass yet on pivot, advance
              one of the preceding terms */
21.        aterm ← pickTerm(terms[0..pTerm])
22.        posting[aterm] ← aterm.iterator.next(pivot)
23.    end repeat
```

pivotnya dapet yang pertama karena 1.6 > 0

hujan adalah pivot term

[hujan, 6, 1.6]	D1, 1.5	D2, 0.4	D3, 0.6	D6, 1.0	D8, 1.5	D11, 1.6	last
[turun, 6, 1.5]	D1, 0.7	D3, 1.0	D6, 1.5	D8, 1.5	D10, 0.3	D12, 1.1	last
[deras, 5, 1.8]	D1, 1.2	D6, 1.0	D7, 0.5	D10, 0.6	D11, 1.8	last	

TOP-1 : []
Threshold : 0
Pivot : 1
curDoc : 0

#documents that are fully evaluated: 0

```
1. Function next(θ)
2.    repeat
3.       /* Sort the terms in non decreasing order of
          DID */
4.       sort(terms, posting)
5.       /* Find pivot term - the first one with accumulated
          UB ≥ θ */
6.       pTerm ← findPivotTerm(terms, θ)
7.       if (pTerm = null) return (NoMoreDocs)
8.       pivot ← posting[pTerm].DID
9.       if (pivot = lastID) return (NoMoreDocs)
10.      if (pivot ≤ curDoc)
11.         /* pivot has already been considered, advance
             one of the preceding terms */
12.         aterm ← pickTerm(terms[0..pTerm])
13.         posting[aterm] ← aterm.iterator.next(curDoc+1)
14.      else /* pivot > curDoc */
15.         if (posting[0].DID = pivot)
16.            /* Success, all terms preceding pTerm belong
                to the pivot */
17.            curDoc ← pivot
18.            return (curDoc, posting)
19.         else
20.            /* not enough mass yet on pivot, advance
                one of the preceding terms */
21.            aterm ← pickTerm(terms[0..pTerm])
22.            posting[aterm] ← aterm.iterator.next(pivot)
23.      end repeat
```

D1 adalah pivot

[hujan, 6, 1.6] →	D1, 1.5	D2, 0.4	D3, 0.6	D6, 1.0	D8, 1.5	D11, 1.6	last
[turun, 6, 1.5] →	D1, 0.7	D3, 1.0	D6, 1.5	D8, 1.5	D10, 0.3	D12, 1.1	last
[deras, 5, 1.8] →	D1, 1.2	D6, 1.0	D7, 0.5	D10, 0.6	D11, 1.8	last	

TOP-1 : [(D1,3.4)]
Threshold : 3.4
Pivot : 1
curDoc : 1

#documents that are fully evaluated: 1

```
1. Function next(θ)
2.   repeat
3.     /* Sort the terms in non decreasing order of
         DID */
4.     sort(terms, posting)
5.     /* Find pivot term - the first one with accumulated
         UB ≥ θ */
6.     pTerm ← findPivotTerm(terms, θ)
7.     if (pTerm = null) return (NoMoreDocs)
8.     pivot ← posting[pTerm].DID
9.     if (pivot = lastID) return (NoMoreDocs)
10.    if (pivot ≤ curDoc)
11.      /* pivot has already been considered, advance
           one of the preceding terms */
12.      aterm ← pickTerm(terms[0..pTerm])
13.      posting[aterm] ← aterm.iterator.next(curDoc+1)
14.    else /* pivot > curDoc */
15.      if (posting[0].DID = pivot)
16.        /* Success, all terms preceding pTerm belong
             to the pivot */
17.        curDoc ← pivot
18.        return (curDoc, posting)
19.      else
20.        /* not enough mass yet on pivot, advance
             one of the preceding terms */
21.        aterm ← pickTerm(terms[0..pTerm])
22.        posting[aterm] ← aterm.iterator.next(pivot)
23.    end repeat
```

artinya ini dievaluasi karena mengalahkan threshold

D1 dikembalikan, dan dihitung score full-nya!
Lalu disimpan di daftar TOP-1 kita, dan jangan lupa update nilai threshold terbaru

WAND(D1) = WAND(hujan, 1.6, turun, 1.5, deras, 1.8; threshold) = True

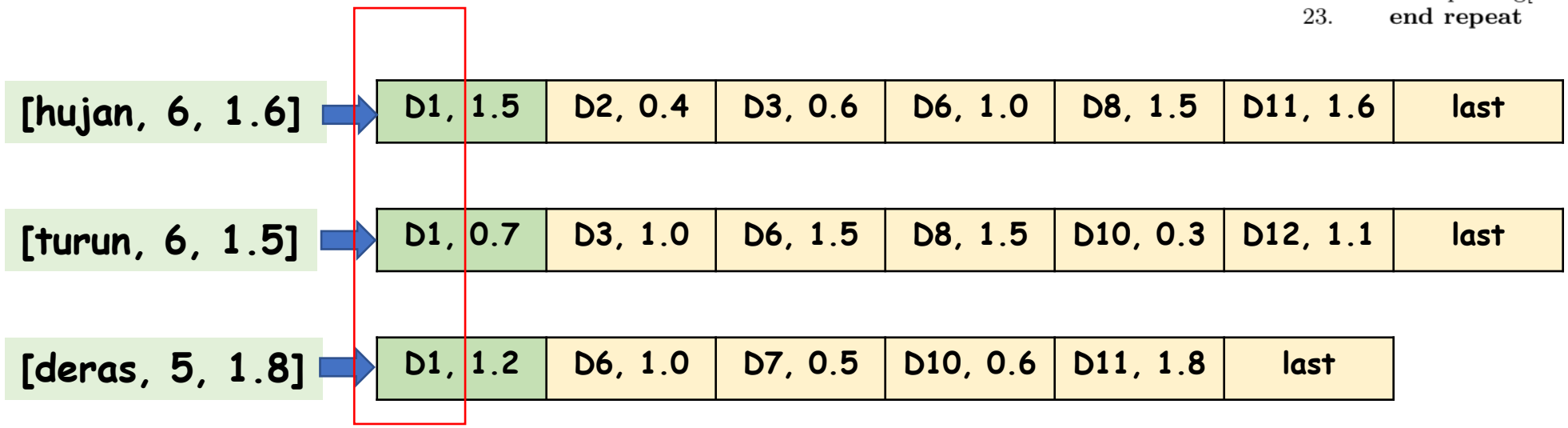
[hujan, 6, 1.6] →	D1, 1.5	D2, 0.4	D3, 0.6	D6, 1.0	D8, 1.5	D11, 1.6	last
[turun, 6, 1.5] →	D1, 0.7	D3, 1.0	D6, 1.5	D8, 1.5	D10, 0.3	D12, 1.1	last
[deras, 5, 1.8] →	D1, 1.2	D6, 1.0	D7, 0.5	D10, 0.6	D11, 1.8	last	

TOP-1 : [(D1,3.4)]
Threshold : 3.4
Pivot : 1
curDoc : 1

#documents that are fully evaluated: 1

```
1. Function next(θ)
2.    repeat
3.       /* Sort the terms in non decreasing order of
          DID */
4.       sort(terms, posting)
5.       /* Find pivot term - the first one with accumulated
          UB ≥ θ */
6.       pTerm ← findPivotTerm(terms, θ)
7.       if (pTerm = null) return (NoMoreDocs)
8.       pivot ← posting[pTerm].DID
9.       if (pivot = lastID) return (NoMoreDocs)
10.      if (pivot ≤ curDoc)
11.         /* pivot has already been considered, advance
             one of the preceding terms */
12.         aterm ← pickTerm(terms[0..pTerm])
13.         posting[aterm] ← aterm.iterator.next(curDoc+1)
14.      else /* pivot > curDoc */
15.         if (posting[0].DID = pivot)
16.            /* Success, all terms preceding pTerm belong
                to the pivot */
17.            curDoc ← pivot
18.            return (curDoc, posting)
19.         else
20.            /* not enough mass yet on pivot, advance
                one of the preceding terms */
21.            aterm ← pickTerm(terms[0..pTerm])
22.            posting[aterm] ← aterm.iterator.next(pivot)
23.      end repeat
```

Mulai dari awal lagi, dan lakukan sort terhadap Doc ID



TOP-1 : [(D1,3.4)]
Threshold : 3.4
Pivot : 1
curDoc : 1

#documents that are fully evaluated: 1

```
1. Function next(θ)
2.   repeat
3.     /* Sort the terms in non decreasing order of
         DID */
4.     sort(terms, posting)
5.     /* Find pivot term - the first one with accumulated
         UB ≥ θ */
6.     pTerm ← findPivotTerm(terms, θ)
7.     if (pTerm = null) return (NoMoreDocs)
8.     pivot ← posting[pTerm].DID
9.     if (pivot = lastID) return (NoMoreDocs)
10.    if (pivot ≤ curDoc)
11.      /* pivot has already been considered, advance
           one of the preceding terms */
12.      aterm ← pickTerm(terms[0..pTerm])
13.      posting[aterm] ← aterm.iterator.next(curDoc+1)
14.    else /* pivot > curDoc */
15.      if (posting[0].DID = pivot)
16.        /* Success, all terms preceding pTerm belong
             to the pivot */
17.        curDoc ← pivot
18.        return (curDoc, posting)
19.      else
20.        /* not enough mass yet on pivot, advance
             one of the preceding terms */
21.        aterm ← pickTerm(terms[0..pTerm])
22.        posting[aterm] ← aterm.iterator.next(pivot)
23.    end repeat
```

"deras" adalah pivot term, dengan D1 adalah pivot

1.8 jadi pivot karena yang punya chance pivot itu adalah 1.6 + 1.5 + X

X Ini adalah 1.8 karena lebih dari sama dengna pivot maka dia jadi pivot term

[hujan, 6, 1.6]	D1, 1.5	D2, 0.4	D3, 0.6	D6, 1.0	D8, 1.5	D11, 1.6	last
[turun, 6, 1.5]	D1, 0.7	D3, 1.0	D6, 1.5	D8, 1.5	D10, 0.3	D12, 1.1	last
[deras, 5, 1.8]	D1, 1.2	D6, 1.0	D7, 0.5	D10, 0.6	D11, 1.8	last	

TOP-1 : [(D1,3.4)]
Threshold : 3.4
Pivot : 1
curDoc : 1

#documents that are fully evaluated: 1

```
1. Function next(θ)
2.   repeat
3.     /* Sort the terms in non decreasing order of
         DID */
4.     sort(terms, posting)
5.     /* Find pivot term - the first one with accumulated
         UB ≥ θ */
6.     pTerm ← findPivotTerm(terms, θ)
7.     if (pTerm = null) return (NoMoreDocs)
8.     pivot ← posting[pTerm].DID
9.     if (pivot = lastID) return (NoMoreDocs)
10.    if (pivot ≤ curDoc)
11.      /* pivot has already been considered, advance
           one of the preceding terms */
12.      aterm ← pickTerm(terms[0..pTerm])
13.      posting[aterm] ← aterm.iterator.next(curDoc+1)
14.    else /* pivot > curDoc */
15.      if (posting[0].DID = pivot)
16.        /* Success, all terms preceding pTerm belong
             to the pivot */
17.        curDoc ← pivot
18.        return (curDoc, posting)
19.      else
20.        /* not enough mass yet on pivot, advance
             one of the preceding terms */
21.        aterm ← pickTerm(terms[0..pTerm])
22.        posting[aterm] ← aterm.iterator.next(pivot)
23.    end repeat
```

Advance one of the preceding terms

pickTerm() → banyak cara untuk memilih; yang paling simple adalah "pilih yang pertama"

Pointer dari term yang terpilih dimajukan hingga bertemu DID pertama yang >= 1 + 1 = 2

[hujan, 6, 1.6] →	D1, 1.5	D2, 0.4	D3, 0.6	D6, 1.0	D8, 1.5	D11, 1.6	last
[turun, 6, 1.5] →	D1, 0.7	D3, 1.0	D6, 1.5	D8, 1.5	D10, 0.3	D12, 1.1	last
[deras, 5, 1.8] →	D1, 1.2	D6, 1.0	D7, 0.5	D10, 0.6	D11, 1.8	last	

TOP-1 : [(D1,3.4)]
Threshold : 3.4
Pivot : 1
curDoc : 1

#documents that are fully evaluated: 1

Mulai lagi dari awal. Sort berdasarkan Doc ID

```
1. Function next(θ)
2.   repeat
3.     /* Sort the terms in non decreasing order of
         DID */
4.     sort(terms, posting)
5.     /* Find pivot term - the first one with accumulated
         UB ≥ θ */
6.     pTerm ← findPivotTerm(terms, θ)
7.     if (pTerm = null) return (NoMoreDocs)
8.     pivot ← posting[pTerm].DID
9.     if (pivot = lastID) return (NoMoreDocs)
10.    if (pivot ≤ curDoc)
11.      /* pivot has already been considered, advance
           one of the preceding terms */
12.      aterm ← pickTerm(terms[0..pTerm])
13.      posting[aterm] ← aterm.iterator.next(curDoc+1)
14.    else /* pivot > curDoc */
15.      if (posting[0].DID = pivot)
16.        /* Success, all terms preceding pTerm belong
             to the pivot */
17.        curDoc ← pivot
18.        return (curDoc, posting)
19.      else
20.        /* not enough mass yet on pivot, advance
             one of the preceding terms */
21.        aterm ← pickTerm(terms[0..pTerm])
22.        posting[aterm] ← aterm.iterator.next(pivot)
23.    end repeat
```

[hujan, 6, 1.6] →	D1, 1.5	D2, 0.4	D3, 0.6	D6, 1.0	D8, 1.5	D11, 1.6	last
[turun, 6, 1.5] →	D1, 0.7	D3, 1.0	D6, 1.5	D8, 1.5	D10, 0.3	D12, 1.1	last
[deras, 5, 1.8] →	D1, 1.2	D6, 1.0	D7, 0.5	D10, 0.6	D11, 1.8	last	

TOP-1 : [(D1,3.4)]
Threshold : 3.4
Pivot : 1
curDoc : 1

#documents that are fully evaluated: 1

Mulai lagi dari awal. Sort berdasarkan Doc ID

```
1. Function next(θ)
2.   repeat
3.     /* Sort the terms in non decreasing order of
          DID */
4.     sort(terms, posting)
5.     /* Find pivot term - the first one with accumulated
          UB ≥ θ */
6.     pTerm ← findPivotTerm(terms, θ)
7.     if (pTerm = null) return (NoMoreDocs)
8.     pivot ← posting[pTerm].DID
9.     if (pivot = lastID) return (NoMoreDocs)
10.    if (pivot ≤ curDoc)
11.      /* pivot has already been considered, advance
            one of the preceding terms */
12.      aterm ← pickTerm(terms[0..pTerm])
13.      posting[aterm] ← aterm.iterator.next(curDoc+1)
14.    else /* pivot > curDoc */
15.      if (posting[0].DID = pivot)
16.        /* Success, all terms preceding pTerm belong
              to the pivot */
17.        curDoc ← pivot
18.        return (curDoc, posting)
19.      else
20.        /* not enough mass yet on pivot, advance
              one of the preceding terms */
21.        aterm ← pickTerm(terms[0..pTerm])
22.        posting[aterm] ← aterm.iterator.next(pivot)
23.    end repeat
```

[turun, 6, 1.5] →	D1, 0.7	D3, 1.0	D6, 1.5	D8, 1.5	D10, 0.3	D12, 1.1	last
[deras, 5, 1.8] →	D1, 1.2	D6, 1.0	D7, 0.5	D10, 0.6	D11, 1.8	last	
[hujan, 6, 1.6] →	D1, 1.5	D2, 0.4	D3, 0.6	D6, 1.0	D8, 1.5	D11, 1.6	last

TOP-1 : [(D1,3.4)]
Threshold : 3.4
Pivot : 2
curDoc : 1

#documents that are fully evaluated: 1

```
1. Function next(θ)
2.    repeat
3.       /* Sort the terms in non decreasing order of
          DID */
4.       sort(terms, posting)
5.       /* Find pivot term - the first one with accumulated
          UB ≥ θ */
6.       pTerm ← findPivotTerm(terms, θ)
7.       if (pTerm = null) return (NoMoreDocs)
8.       pivot ← posting[pTerm].DID
9.       if (pivot = lastID) return (NoMoreDocs)
10.      if (pivot ≤ curDoc)
11.         /* pivot has already been considered, advance
             one of the preceding terms */
12.         aterm ← pickTerm(terms[0..pTerm])
13.         posting[aterm] ← aterm.iterator.next(curDoc+1)
14.      else /* pivot > curDoc */
15.         if (posting[0].DID = pivot)
16.            /* Success, all terms preceding pTerm belong
                to the pivot */
17.            curDoc ← pivot
18.            return (curDoc, posting)
19.         else
20.            /* not enough mass yet on pivot, advance
                one of the preceding terms */
21.            aterm ← pickTerm(terms[0..pTerm])
22.            posting[aterm] ← aterm.iterator.next(pivot)
23.      end repeat
```

Mulai lagi dari awal. Sort berdasarkan Doc ID

1.5 + 1.8 + 1.6 >= 3.4
Pivot term = hujan, pivot = D2

[turun, 6, 1.5] →	D1, 0.7	D3, 1.0	D6, 1.5	D8, 1.5	D10, 0.3	D12, 1.1	last
[deras, 5, 1.8] →	D1, 1.2	D6, 1.0	D7, 0.5	D10, 0.6	D11, 1.8	last	
[hujan, 6, 1.6] →	D1, 1.5	D2, 0.4	D3, 0.6	D6, 1.0	D8, 1.5	D11, 1.6	last

TOP-1 : [(D1,3.4)]
Threshold : 3.4
Pivot : 2
curDoc : 1

#documents that are fully evaluated: 1

```
1. Function next(θ)
2.   repeat
3.     /* Sort the terms in non decreasing order of
         DID */
4.     sort(terms, posting)
5.     /* Find pivot term - the first one with accumulated
         UB ≥ θ */
6.     pTerm ← findPivotTerm(terms, θ)
7.     if (pTerm = null) return (NoMoreDocs)
8.     pivot ← posting[pTerm].DID
9.     if (pivot = lastID) return (NoMoreDocs)
10.    if (pivot ≤ curDoc)
11.      /* pivot has already been considered, advance
           one of the preceding terms */
12.      aterm ← pickTerm(terms[0..pTerm])
13.      posting[aterm] ← aterm.iterator.next(curDoc+1)
14.    else /* pivot > curDoc */
15.      if (posting[0].DID = pivot)
16.        /* Success, all terms preceding pTerm belong
             to the pivot */
17.        curDoc ← pivot
18.        return (curDoc, posting)
19.      else
20.        /* not enough mass yet on pivot, advance
             one of the preceding terms */
21.        aterm ← pickTerm(terms[0..pTerm])
22.        posting[aterm] ← aterm.iterator.next(pivot)
23.    end repeat
```

Not enough mass yet on pivot, advance one of the preceding terms.

pickTerm() → kita pilih yang pertama

[turun, 6, 1.5] →	D1, 0.7	D3, 1.0	D6, 1.5	D8, 1.5	D10, 0.3	D12, 1.1	last
[deras, 5, 1.8] →	D1, 1.2	D6, 1.0	D7, 0.5	D10, 0.6	D11, 1.8	last	
[hujan, 6, 1.6] →	D1, 1.5	D2, 0.4	D3, 0.6	D6, 1.0	D8, 1.5	D11, 1.6	last

TOP-1 : [(D1,3.4)]
Threshold : 3.4
Pivot : 2
curDoc : 1

#documents that are fully evaluated: 1

```
1. Function next(θ)
2.    repeat
3.       /* Sort the terms in non decreasing order of
          DID */
4.       sort(terms, posting)
5.       /* Find pivot term - the first one with accumulated
          UB ≥ θ */
6.       pTerm ← findPivotTerm(terms, θ)
7.       if (pTerm = null) return (NoMoreDocs)
8.       pivot ← posting[pTerm].DID
9.       if (pivot = lastID) return (NoMoreDocs)
10.      if (pivot ≤ curDoc)
11.         /* pivot has already been considered, advance
             one of the preceding terms */
12.         aterm ← pickTerm(terms[0..pTerm])
13.         posting[aterm] ← aterm.iterator.next(curDoc+1)
14.      else /* pivot > curDoc */
15.         if (posting[0].DID = pivot)
16.            /* Success, all terms preceding pTerm belong
                to the pivot */
17.            curDoc ← pivot
18.            return (curDoc, posting)
19.         else
20.            /* not enough mass yet on pivot, advance
                one of the preceding terms */
21.            aterm ← pickTerm(terms[0..pTerm])
22.            posting[aterm] ← aterm.iterator.next(pivot)
23.   end repeat
```

Not enough mass yet on pivot, advance one of the preceding terms.

pickTerm() → kita pilih yang pertama
Lalu kita pindahkan ke posisi pertama yang Doc ID >= pivot

[turun, 6, 1.5]	D1, 0.7	D3, 1.0	D6, 1.5	D8, 1.5	D10, 0.3	D12, 1.1	last
[deras, 5, 1.8]	D1, 1.2	D6, 1.0	D7, 0.5	D10, 0.6	D11, 1.8	last	
[hujan, 6, 1.6]	D1, 1.5	D2, 0.4	D3, 0.6	D6, 1.0	D8, 1.5	D11, 1.6	last

TOP-1 : [(D1,3.4)]
Threshold : 3.4
Pivot : 2
curDoc : 1

#documents that are fully evaluated: 1

Mulai lagi dari awal, lakukan sorting

```
1. Function next(θ)
2.   repeat
3.     /* Sort the terms in non decreasing order of
          DID */
4.     sort(terms, posting)
5.     /* Find pivot term - the first one with accumulated
          UB ≥ θ */
6.     pTerm ← findPivotTerm(terms, θ)
7.     if (pTerm = null) return (NoMoreDocs)
8.     pivot ← posting[pTerm].DID
9.     if (pivot = lastID) return (NoMoreDocs)
10.    if (pivot ≤ curDoc)
11.      /* pivot has already been considered, advance
            one of the preceding terms */
12.      aterm ← pickTerm(terms[0..pTerm])
13.      posting[aterm] ← aterm.iterator.next(curDoc+1)
14.    else /* pivot > curDoc */
15.      if (posting[0].DID = pivot)
16.        /* Success, all terms preceding pTerm belong
              to the pivot */
17.        curDoc ← pivot
18.        return (curDoc, posting)
19.      else
20.        /* not enough mass yet on pivot, advance
              one of the preceding terms */
21.        aterm ← pickTerm(terms[0..pTerm])
22.        posting[aterm] ← aterm.iterator.next(pivot)
23.    end repeat
```

[turun, 6, 1.5] →	D1, 0.7	D3, 1.0	D6, 1.5	D8, 1.5	D10, 0.3	D12, 1.1	last
[deras, 5, 1.8] →	D1, 1.2	D6, 1.0	D7, 0.5	D10, 0.6	D11, 1.8	last	
[hujan, 6, 1.6] →	D1, 1.5	D2, 0.4	D3, 0.6	D6, 1.0	D8, 1.5	D11, 1.6	last

TOP-1 : [(D1,3.4)]
Threshold : 3.4
Pivot : 2
curDoc : 1

#documents that are fully evaluated: 1

Mulai lagi dari awal, lakukan sorting

```
1. Function next(θ)
2.   repeat
3.     /* Sort the terms in non decreasing order of
          DID */
4.     sort(terms, posting)
5.     /* Find pivot term - the first one with accumulated
          UB ≥ θ */
6.     pTerm ← findPivotTerm(terms, θ)
7.     if (pTerm = null) return (NoMoreDocs)
8.     pivot ← posting[pTerm].DID
9.     if (pivot = lastID) return (NoMoreDocs)
10.    if (pivot ≤ curDoc)
11.      /* pivot has already been considered, advance
            one of the preceding terms */
12.      aterm ← pickTerm(terms[0..pTerm])
13.      posting[aterm] ← aterm.iterator.next(curDoc+1)
14.    else /* pivot > curDoc */
15.      if (posting[0].DID = pivot)
16.        /* Success, all terms preceding pTerm belong
              to the pivot */
17.        curDoc ← pivot
18.        return (curDoc, posting)
19.      else
20.        /* not enough mass yet on pivot, advance
              one of the preceding terms */
21.        aterm ← pickTerm(terms[0..pTerm])
22.        posting[aterm] ← aterm.iterator.next(pivot)
23.    end repeat
```

[deras, 5, 1.8] →	D1, 1.2	D6, 1.0	D7, 0.5	D10, 0.6	D11, 1.8	last	
[hujan, 6, 1.6] →	D1, 1.5	D2, 0.4	D3, 0.6	D6, 1.0	D8, 1.5	D11, 1.6	last
[turun, 6, 1.5] →	D1, 0.7	D3, 1.0	D6, 1.5	D8, 1.5	D10, 0.3	D12, 1.1	last

TOP-1 : [(D1,3.4)]
Threshold : 3.4
Pivot : 2
curDoc : 1

#documents that are fully evaluated: 1

Pivot term = hujan dengan pivot = D2

```
1. Function next(θ)
2.    repeat
3.       /* Sort the terms in non decreasing order of
          DID */
4.       sort(terms, posting)
5.       /* Find pivot term - the first one with accumulated
          UB ≥ θ */
6.       pTerm ← findPivotTerm(terms, θ)
7.       if (pTerm = null) return (NoMoreDocs)
8.       pivot ← posting[pTerm].DID
9.       if (pivot = lastID) return (NoMoreDocs)
10.      if (pivot ≤ curDoc)
11.         /* pivot has already been considered, advance
             one of the preceding terms */
12.         aterm ← pickTerm(terms[0..pTerm])
13.         posting[aterm] ← aterm.iterator.next(curDoc+1)
14.      else /* pivot > curDoc */
15.         if (posting[0].DID = pivot)
16.            /* Success, all terms preceding pTerm belong
                to the pivot */
17.            curDoc ← pivot
18.            return (curDoc, posting)
19.         else
20.            /* not enough mass yet on pivot, advance
                one of the preceding terms */
21.            aterm ← pickTerm(terms[0..pTerm])
22.            posting[aterm] ← aterm.iterator.next(pivot)
23.      end repeat
```

[deras, 5, 1.8] →	D1, 1.2	D6, 1.0	D7, 0.5	D10, 0.6	D11, 1.8	last	
[hujan, 6, 1.6] →	D1, 1.5	D2, 0.4	D3, 0.6	D6, 1.0	D8, 1.5	D11, 1.6	last
[turun, 6, 1.5] →	D1, 0.7	D3, 1.0	D6, 1.5	D8, 1.5	D10, 0.3	D12, 1.1	last

TOP-1 : [(D1,3.4)]
Threshold : 3.4
Pivot : 2
curDoc : 1

#documents that are fully evaluated: 1

```
1. Function next(θ)
2.   repeat
3.     /* Sort the terms in non decreasing order of
         DID */
4.     sort(terms, posting)
5.     /* Find pivot term - the first one with accumulated
         UB ≥ θ */
6.     pTerm ← findPivotTerm(terms, θ)
7.     if (pTerm = null) return (NoMoreDocs)
8.     pivot ← posting[pTerm].DID
9.     if (pivot = lastID) return (NoMoreDocs)
10.    if (pivot ≤ curDoc)
11.      /* pivot has already been considered, advance
           one of the preceding terms */
12.      aterm ← pickTerm(terms[0..pTerm])
13.      posting[aterm] ← aterm.iterator.next(curDoc+1)
14.    else /* pivot > curDoc */
15.      if (posting[0].DID = pivot)
16.        /* Success, all terms preceding pTerm belong
             to the pivot */
17.        curDoc ← pivot
18.        return (curDoc, posting)
19.      else
20.        /* not enough mass yet on pivot, advance
             one of the preceding terms */
21.        aterm ← pickTerm(terms[0..pTerm])
22.        posting[aterm] ← aterm.iterator.next(pivot)
23.    end repeat
```

Kita majukan pointer di deras ke Doc ID pertama yang >= pivot.

[deras, 5, 1.8] →	D1, 1.2	D6, 1.0	D7, 0.5	D10, 0.6	D11, 1.8	last

[hujan, 6, 1.6] →	D1, 1.5	D2, 0.4	D3, 0.6	D6, 1.0	D8, 1.5	D11, 1.6	last

[turun, 6, 1.5] →	D1, 0.7	D3, 1.0	D6, 1.5	D8, 1.5	D10, 0.3	D12, 1.1	last

TOP-1 : [(D1,3.4)]
Threshold : 3.4
Pivot : 2
curDoc : 1

#documents that are fully evaluated: 1

```
1. Function next(θ)
2.    repeat
3.       /* Sort the terms in non decreasing order of
          DID */
4.       sort(terms, posting)
5.       /* Find pivot term - the first one with accumulated
          UB ≥ θ */
6.       pTerm ← findPivotTerm(terms, θ)
7.       if (pTerm = null) return (NoMoreDocs)
8.       pivot ← posting[pTerm].DID
9.       if (pivot = lastID) return (NoMoreDocs)
10.      if (pivot ≤ curDoc)
11.         /* pivot has already been considered, advance
             one of the preceding terms */
12.         aterm ← pickTerm(terms[0..pTerm])
13.         posting[aterm] ← aterm.iterator.next(curDoc+1)
14.      else /* pivot > curDoc */
15.         if (posting[0].DID = pivot)
16.            /* Success, all terms preceding pTerm belong
                to the pivot */
17.            curDoc ← pivot
18.            return (curDoc, posting)
19.         else
20.            /* not enough mass yet on pivot, advance
                one of the preceding terms */
21.            aterm ← pickTerm(terms[0..pTerm])
22.            posting[aterm] ← aterm.iterator.next(pivot)
23.      end repeat
```

Kita majukan pointer di deras ke Doc ID pertama yang >= pivot.

[deras, 5, 1.8] →	D1, 1.2	D6, 1.0	D7, 0.5	D10, 0.6	D11, 1.8	last	
[hujan, 6, 1.6] →	D1, 1.5	D2, 0.4	D3, 0.6	D6, 1.0	D8, 1.5	D11, 1.6	last
[turun, 6, 1.5] →	D1, 0.7	D3, 1.0	D6, 1.5	D8, 1.5	D10, 0.3	D12, 1.1	last

TOP-1 : [(D1,3.4)]
Threshold : 3.4
Pivot : 2
curDoc : 1

#documents that are fully evaluated: 1

Sort berdasarkan Doc ID

```
1. Function next(θ)
2.    repeat
3.       /* Sort the terms in non decreasing order of
          DID */
4.       sort(terms, posting)
5.       /* Find pivot term - the first one with accumulated
          UB ≥ θ */
6.       pTerm ← findPivotTerm(terms, θ)
7.       if (pTerm = null) return (NoMoreDocs)
8.       pivot ← posting[pTerm].DID
9.       if (pivot = lastID) return (NoMoreDocs)
10.      if (pivot ≤ curDoc)
11.         /* pivot has already been considered, advance
             one of the preceding terms */
12.         aterm ← pickTerm(terms[0..pTerm])
13.         posting[aterm] ← aterm.iterator.next(curDoc+1)
14.      else /* pivot > curDoc */
15.         if (posting[0].DID = pivot)
16.            /* Success, all terms preceding pTerm belong
                to the pivot */
17.            curDoc ← pivot
18.            return (curDoc, posting)
19.         else
20.            /* not enough mass yet on pivot, advance
                one of the preceding terms */
21.            aterm ← pickTerm(terms[0..pTerm])
22.            posting[aterm] ← aterm.iterator.next(pivot)
23.      end repeat
```

[hujan, 6, 1.6] →	D1, 1.5	D2, 0.4	D3, 0.6	D6, 1.0	D8, 1.5	D11, 1.6	last
[turun, 6, 1.5] →	D1, 0.7	D3, 1.0	D6, 1.5	D8, 1.5	D10, 0.3	D12, 1.1	last
[deras, 5, 1.8] →	D1, 1.2	D6, 1.0	D7, 0.5	D10, 0.6	D11, 1.8	last	

TOP-1 : [(D1,3.4)]
Threshold : 3.4
Pivot : 6
curDoc : 1

#documents that are fully evaluated: 1

Pivot term = deras, dengan pivot = D6

```
1. Function next(θ)
2.    repeat
3.       /* Sort the terms in non decreasing order of
          DID */
4.       sort(terms, posting)
5.       /* Find pivot term - the first one with accumulated
          UB ≥ θ */
6.       pTerm ← findPivotTerm(terms, θ)
7.       if (pTerm = null) return (NoMoreDocs)
8.       pivot ← posting[pTerm].DID
9.       if (pivot = lastID) return (NoMoreDocs)
10.      if (pivot ≤ curDoc)
11.         /* pivot has already been considered, advance
             one of the preceding terms */
12.         aterm ← pickTerm(terms[0..pTerm])
13.         posting[aterm] ← aterm.iterator.next(curDoc+1)
14.      else /* pivot > curDoc */
15.         if (posting[0].DID = pivot)
16.            /* Success, all terms preceding pTerm belong
                to the pivot */
17.            curDoc ← pivot
18.            return (curDoc, posting)
19.         else
20.            /* not enough mass yet on pivot, advance
                one of the preceding terms */
21.            aterm ← pickTerm(terms[0..pTerm])
22.            posting[aterm] ← aterm.iterator.next(pivot)
23.      end repeat
```

[hujan, 6, 1.6] →	D1, 1.5	D2, 0.4	D3, 0.6	D6, 1.0	D8, 1.5	D11, 1.6	last
[turun, 6, 1.5] →	D1, 0.7	D3, 1.0	D6, 1.5	D8, 1.5	D10, 0.3	D12, 1.1	last
[deras, 5, 1.8] →	D1, 1.2	D6, 1.0	D7, 0.5	D10, 0.6	D11, 1.8	last	

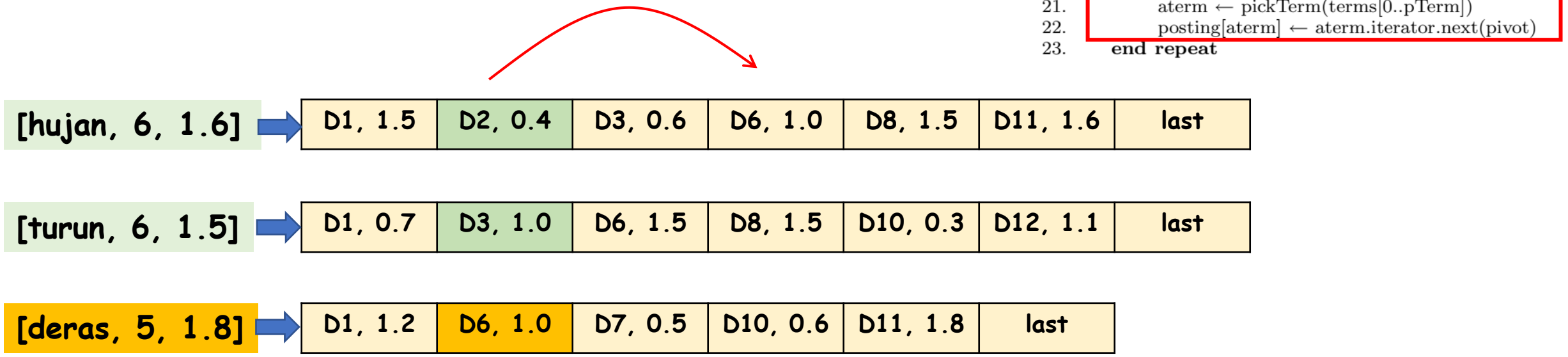
TOP-1 : [(D1,3.4)]
Threshold : 3.4
Pivot : 6
curDoc : 1

#documents that are fully evaluated: 1

Not enough mass yet on pivot, advance one of the preceding terms.

pickTerm() → kita pilih yang pertama
Lalu kita pindahkan ke posisi pertama yang Doc ID >= pivot

```
1. Function next(θ)
2.   repeat
3.     /* Sort the terms in non decreasing order of
          DID */
4.     sort(terms, posting)
5.     /* Find pivot term - the first one with accumulated
          UB ≥ θ */
6.     pTerm ← findPivotTerm(terms, θ)
7.     if (pTerm = null) return (NoMoreDocs)
8.     pivot ← posting[pTerm].DID
9.     if (pivot = lastID) return (NoMoreDocs)
10.    if (pivot ≤ curDoc)
11.      /* pivot has already been considered, advance
            one of the preceding terms */
12.      aterm ← pickTerm(terms[0..pTerm])
13.      posting[aterm] ← aterm.iterator.next(curDoc+1)
14.    else /* pivot > curDoc */
15.      if (posting[0].DID = pivot)
16.        /* Success, all terms preceding pTerm belong
              to the pivot */
17.        curDoc ← pivot
18.        return (curDoc, posting)
19.      else
20.        /* not enough mass yet on pivot, advance
              one of the preceding terms */
21.        aterm ← pickTerm(terms[0..pTerm])
22.        posting[aterm] ← aterm.iterator.next(pivot)
23.    end repeat
```



TOP-1 : [(D1,3.4)]
Threshold : 3.4
Pivot : 6
curDoc : 1

#documents that are fully evaluated: 1

D3 di-skip karena dijamin tidak akan memberikan score yang > 3.4. Mengapa? ☺

Catat bahwa posting list terakhir berada pada posisi D6. Seandainya dua posting list pertama berada pada posisi D3, paling tinggi hanya 1.6 + 1.5 = 3.1 yang < 3.4.

```
1. Function next(θ)
2.   repeat
3.     /* Sort the terms in non decreasing order of
          DID */
4.     sort(terms, posting)
5.     /* Find pivot term - the first one with accumulated
          UB ≥ θ */
6.     pTerm ← findPivotTerm(terms, θ)
7.     if (pTerm = null) return (NoMoreDocs)
8.     pivot ← posting[pTerm].DID
9.     if (pivot = lastID) return (NoMoreDocs)
10.    if (pivot ≤ curDoc)
11.      /* pivot has already been considered, advance
            one of the preceding terms */
12.      aterm ← pickTerm(terms[0..pTerm])
13.      posting[aterm] ← aterm.iterator.next(curDoc+1)
14.    else /* pivot > curDoc */
15.      if (posting[0].DID = pivot)
16.        /* Success, all terms preceding pTerm belong
              to the pivot */
17.        curDoc ← pivot
18.        return (curDoc, posting)
19.      else
20.        /* not enough mass yet on pivot, advance
              one of the preceding terms */
21.        aterm ← pickTerm(terms[0..pTerm])
22.        posting[aterm] ← aterm.iterator.next(pivot)
23.    end repeat
```

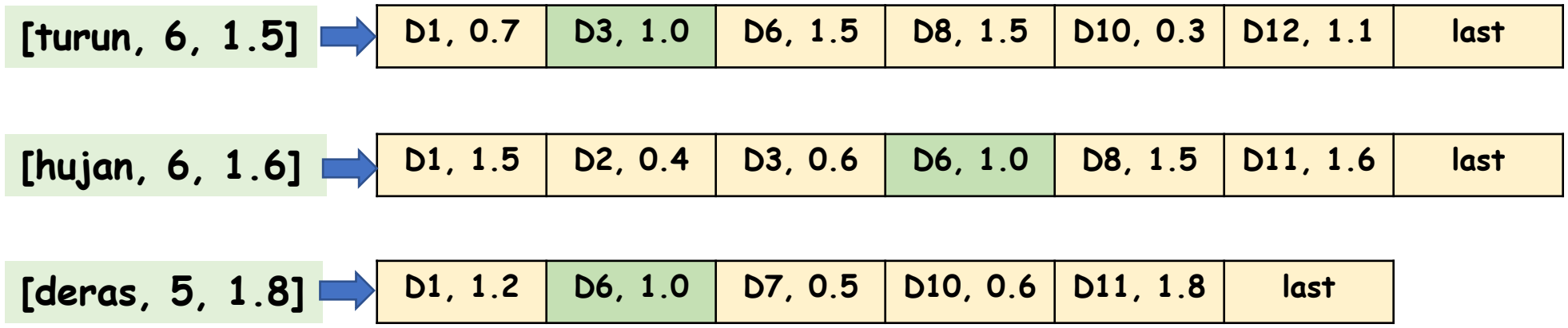
[hujan, 6, 1.6] →	D1, 1.5	D2, 0.4	D3, 0.6	D6, 1.0	D8, 1.5	D11, 1.6	last
[turun, 6, 1.5] →	D1, 0.7	D3, 1.0	D6, 1.5	D8, 1.5	D10, 0.3	D12, 1.1	last
[deras, 5, 1.8] →	D1, 1.2	D6, 1.0	D7, 0.5	D10, 0.6	D11, 1.8	last	

TOP-1 : [(D1,3.4)]
Threshold : 3.4
Pivot : 6
curDoc : 1

#documents that are fully evaluated: 1

Sort dokumen berdasarkan Doc ID

```
1. Function next(θ)
2.    repeat
3.       /* Sort the terms in non decreasing order of
          DID */
4.       sort(terms, posting)
5.       /* Find pivot term - the first one with accumulated
          UB ≥ θ */
6.       pTerm ← findPivotTerm(terms, θ)
7.       if (pTerm = null) return (NoMoreDocs)
8.       pivot ← posting[pTerm].DID
9.       if (pivot = lastID) return (NoMoreDocs)
10.      if (pivot ≤ curDoc)
11.         /* pivot has already been considered, advance
             one of the preceding terms */
12.         aterm ← pickTerm(terms[0..pTerm])
13.         posting[aterm] ← aterm.iterator.next(curDoc+1)
14.      else /* pivot > curDoc */
15.         if (posting[0].DID = pivot)
16.            /* Success, all terms preceding pTerm belong
                to the pivot */
17.            curDoc ← pivot
18.            return (curDoc, posting)
19.         else
20.            /* not enough mass yet on pivot, advance
                one of the preceding terms */
21.            aterm ← pickTerm(terms[0..pTerm])
22.            posting[aterm] ← aterm.iterator.next(pivot)
23.      end repeat
```



TOP-1 : [(D1,3.4)]
Threshold : 3.4
Pivot : 6
curDoc : 1

#documents that are fully evaluated: 1

Pivot term = deras, dengan pivot = 6

```
1. Function next(θ)
2.   repeat
3.     /* Sort the terms in non decreasing order of
         DID */
4.     sort(terms, posting)
5.     /* Find pivot term - the first one with accumulated
         UB ≥ θ */
6.     pTerm ← findPivotTerm(terms, θ)
7.     if (pTerm = null) return (NoMoreDocs)
8.     pivot ← posting[pTerm].DID
9.     if (pivot = lastID) return (NoMoreDocs)
10.    if (pivot ≤ curDoc)
11.      /* pivot has already been considered, advance
           one of the preceding terms */
12.      aterm ← pickTerm(terms[0..pTerm])
13.      posting[aterm] ← aterm.iterator.next(curDoc+1)
14.    else /* pivot > curDoc */
15.      if (posting[0].DID = pivot)
16.        /* Success, all terms preceding pTerm belong
             to the pivot */
17.        curDoc ← pivot
18.        return (curDoc, posting)
19.      else
20.        /* not enough mass yet on pivot, advance
             one of the preceding terms */
21.        aterm ← pickTerm(terms[0..pTerm])
22.        posting[aterm] ← aterm.iterator.next(pivot)
23.    end repeat
```

[turun, 6, 1.5] →	D1, 0.7	D3, 1.0	D6, 1.5	D8, 1.5	D10, 0.3	D12, 1.1	last
[hujan, 6, 1.6] →	D1, 1.5	D2, 0.4	D3, 0.6	D6, 1.0	D8, 1.5	D11, 1.6	last
[deras, 5, 1.8] →	D1, 1.2	D6, 1.0	D7, 0.5	D10, 0.6	D11, 1.8	last	

TOP-1 : [(D1,3.4)]
Threshold : 3.4
Pivot : 6
curDoc : 1

#documents that are fully evaluated: 1

Majukan Doc ID di term turun ke Doc ID pertama yang >= pivot

```
1. Function next(θ)
2.   repeat
3.     /* Sort the terms in non decreasing order of
          DID */
4.     sort(terms, posting)
5.     /* Find pivot term - the first one with accumulated
          UB ≥ θ */
6.     pTerm ← findPivotTerm(terms, θ)
7.     if (pTerm = null) return (NoMoreDocs)
8.     pivot ← posting[pTerm].DID
9.     if (pivot = lastID) return (NoMoreDocs)
10.    if (pivot ≤ curDoc)
11.      /* pivot has already been considered, advance
            one of the preceding terms */
12.      aterm ← pickTerm(terms[0..pTerm])
13.      posting[aterm] ← aterm.iterator.next(curDoc+1)
14.    else /* pivot > curDoc */
15.      if (posting[0].DID = pivot)
16.        /* Success, all terms preceding pTerm belong
              to the pivot */
17.        curDoc ← pivot
18.        return (curDoc, posting)
19.      else
20.        /* not enough mass yet on pivot, advance
              one of the preceding terms */
21.        aterm ← pickTerm(terms[0..pTerm])
22.        posting[aterm] ← aterm.iterator.next(pivot)
23.    end repeat
```

[turun, 6, 1.5] →	D1, 0.7	D3, 1.0	D6, 1.5	D8, 1.5	D10, 0.3	D12, 1.1	last
[hujan, 6, 1.6] →	D1, 1.5	D2, 0.4	D3, 0.6	D6, 1.0	D8, 1.5	D11, 1.6	last
[deras, 5, 1.8] →	D1, 1.2	D6, 1.0	D7, 0.5	D10, 0.6	D11, 1.8	last	

TOP-1 : [(D1,3.4)]
Threshold : 3.4
Pivot : 6
curDoc : 1

#documents that are fully evaluated: 1

Majukan Doc ID di term turun ke Doc ID pertama yang >= pivot

```
1. Function next(θ)
2.   repeat
3.     /* Sort the terms in non decreasing order of
          DID */
4.     sort(terms, posting)
5.     /* Find pivot term - the first one with accumulated
          UB ≥ θ */
6.     pTerm ← findPivotTerm(terms, θ)
7.     if (pTerm = null) return (NoMoreDocs)
8.     pivot ← posting[pTerm].DID
9.     if (pivot = lastID) return (NoMoreDocs)
10.    if (pivot ≤ curDoc)
11.      /* pivot has already been considered, advance
            one of the preceding terms */
12.      aterm ← pickTerm(terms[0..pTerm])
13.      posting[aterm] ← aterm.iterator.next(curDoc+1)
14.    else /* pivot > curDoc */
15.      if (posting[0].DID = pivot)
16.        /* Success, all terms preceding pTerm belong
              to the pivot */
17.        curDoc ← pivot
18.        return (curDoc, posting)
19.      else
20.        /* not enough mass yet on pivot, advance
              one of the preceding terms */
21.        aterm ← pickTerm(terms[0..pTerm])
22.        posting[aterm] ← aterm.iterator.next(pivot)
23.    end repeat
```

[turun, 6, 1.5]	D1, 0.7	D3, 1.0	D6, 1.5	D8, 1.5	D10, 0.3	D12, 1.1	last
[hujan, 6, 1.6]	D1, 1.5	D2, 0.4	D3, 0.6	D6, 1.0	D8, 1.5	D11, 1.6	last
[deras, 5, 1.8]	D1, 1.2	D6, 1.0	D7, 0.5	D10, 0.6	D11, 1.8	last	

TOP-1 : [(D1,3.4)]
Threshold : 3.4
Pivot : 6
curDoc : 1

#documents that are fully evaluated: 1

```
1. Function next(θ)
2.    repeat
3.       /* Sort the terms in non decreasing order of
          DID */
4.       sort(terms, posting)
5.       /* Find pivot term - the first one with accumulated
          UB ≥ θ */
6.       pTerm ← findPivotTerm(terms, θ)
7.       if (pTerm = null) return (NoMoreDocs)
8.       pivot ← posting[pTerm].DID
9.       if (pivot = lastID) return (NoMoreDocs)
10.      if (pivot ≤ curDoc)
11.         /* pivot has already been considered, advance
             one of the preceding terms */
12.         aterm ← pickTerm(terms[0..pTerm])
13.         posting[aterm] ← aterm.iterator.next(curDoc+1)
14.      else /* pivot > curDoc */
15.         if (posting[0].DID = pivot)
16.            /* Success, all terms preceding pTerm belong
                to the pivot */
17.            curDoc ← pivot
18.            return (curDoc, posting)
19.         else
20.            /* not enough mass yet on pivot, advance
                one of the preceding terms */
21.            aterm ← pickTerm(terms[0..pTerm])
22.            posting[aterm] ← aterm.iterator.next(pivot)
23.      end repeat
```

[turun, 6, 1.5] →	D1, 0.7	D3, 1.0	D6, 1.5	D8, 1.5	D10, 0.3	D12, 1.1	last
[hujan, 6, 1.6] →	D1, 1.5	D2, 0.4	D3, 0.6	D6, 1.0	D8, 1.5	D11, 1.6	last
[deras, 5, 1.8] →	D1, 1.2	D6, 1.0	D7, 0.5	D10, 0.6	D11, 1.8	last	

TOP-1 : [(D1,3.4)]
Threshold : 3.4
Pivot : 6
curDoc : 1

#documents that are fully evaluated: 1

Artinya pivot ID muncul di semua preceding lists!

Dokumen pivot dievaluasi full = 1.5 + 1.0 + 1.0 = 3.5

3.5 > 3.4!

```
1. Function next(θ)
2.   repeat
3.     /* Sort the terms in non decreasing order of
          DID */
4.     sort(terms, posting)
5.     /* Find pivot term - the first one with accumulated
          UB ≥ θ */
6.     pTerm ← findPivotTerm(terms, θ)
7.     if (pTerm = null) return (NoMoreDocs)
8.     pivot ← posting[pTerm].DID
9.     if (pivot = lastID) return (NoMoreDocs)
10.    if (pivot ≤ curDoc)
11.      /* pivot has already been considered, advance
            one of the preceding terms */
12.      aterm ← pickTerm(terms[0..pTerm])
13.      posting[aterm] ← aterm.iterator.next(curDoc+1)
14.    else /* pivot > curDoc */
15.      if (posting[0].DID = pivot)
16.        /* Success, all terms preceding pTerm belong
              to the pivot */
17.        curDoc ← pivot
18.        return (curDoc, posting)
19.      else
20.        /* not enough mass yet on pivot, advance
              one of the preceding terms */
21.        aterm ← pickTerm(terms[0..pTerm])
22.        posting[aterm] ← aterm.iterator.next(pivot)
23.    end repeat
```

[turun, 6, 1.5] →	D1, 0.7	D3, 1.0	D6, 1.5	D8, 1.5	D10, 0.3	D12, 1.1	last
[hujan, 6, 1.6] →	D1, 1.5	D2, 0.4	D3, 0.6	D6, 1.0	D8, 1.5	D11, 1.6	last
[deras, 5, 1.8] →	D1, 1.2	D6, 1.0	D7, 0.5	D10, 0.6	D11, 1.8	last	

TOP-1 : [(D6,3.5)]
Threshold : 3.5
Pivot : 6
curDoc : 6

#documents that are fully evaluated: 2

Artinya pivot ID muncul di semua preceding lists!

Dokumen pivot dievaluasi full = 1.5 + 1.0 + 1.0 = 3.5

3.5 > 3.4!

```
1. Function next(θ)
2.   repeat
3.     /* Sort the terms in non decreasing order of
         DID */
4.     sort(terms, posting)
5.     /* Find pivot term - the first one with accumulated
         UB ≥ θ */
6.     pTerm ← findPivotTerm(terms, θ)
7.     if (pTerm = null) return (NoMoreDocs)
8.     pivot ← posting[pTerm].DID
9.     if (pivot = lastID) return (NoMoreDocs)
10.    if (pivot ≤ curDoc)
11.      /* pivot has already been considered, advance
           one of the preceding terms */
12.      aterm ← pickTerm(terms[0..pTerm])
13.      posting[aterm] ← aterm.iterator.next(curDoc+1)
14.    else /* pivot > curDoc */
15.      if (posting[0].DID = pivot)
16.        /* Success, all terms preceding pTerm belong
             to the pivot */
17.        curDoc ← pivot
18.        return (curDoc, posting)
19.      else
20.        /* not enough mass yet on pivot, advance
             one of the preceding terms */
21.        aterm ← pickTerm(terms[0..pTerm])
22.        posting[aterm] ← aterm.iterator.next(pivot)
23.    end repeat
```

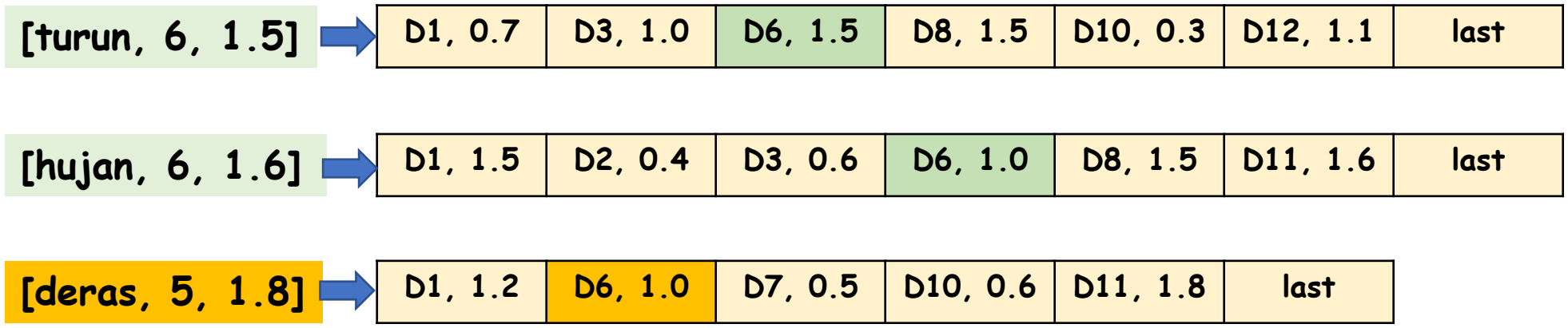
WAND(D6) = WAND(hujan, 1.6, turun, 1.5, deras, 1.8; threshold) = True

[turun, 6, 1.5] →	D1, 0.7	D3, 1.0	D6, 1.5	D8, 1.5	D10, 0.3	D12, 1.1	last
[hujan, 6, 1.6] →	D1, 1.5	D2, 0.4	D3, 0.6	D6, 1.0	D8, 1.5	D11, 1.6	last
[deras, 5, 1.8] →	D1, 1.2	D6, 1.0	D7, 0.5	D10, 0.6	D11, 1.8	last	

TOP-1 : [(D6,3.5)]
Threshold : 3.5
Pivot : 6
curDoc : 6

#documents that are fully evaluated: 2

```
1. Function next(θ)
2.    repeat
3.       /* Sort the terms in non decreasing order of
          DID */
4.       sort(terms, posting)
5.       /* Find pivot term - the first one with accumulated
          UB ≥ θ */
6.       pTerm ← findPivotTerm(terms, θ)
7.       if (pTerm = null) return (NoMoreDocs)
8.       pivot ← posting[pTerm].DID
9.       if (pivot = lastID) return (NoMoreDocs)
10.      if (pivot ≤ curDoc)
11.         /* pivot has already been considered, advance
             one of the preceding terms */
12.         aterm ← pickTerm(terms[0..pTerm])
13.         posting[aterm] ← aterm.iterator.next(curDoc+1)
14.      else /* pivot > curDoc */
15.         if (posting[0].DID = pivot)
16.            /* Success, all terms preceding pTerm belong
                to the pivot */
17.            curDoc ← pivot
18.            return (curDoc, posting)
19.         else
20.            /* not enough mass yet on pivot, advance
                one of the preceding terms */
21.            aterm ← pickTerm(terms[0..pTerm])
22.            posting[aterm] ← aterm.iterator.next(pivot)
23.      end repeat
```



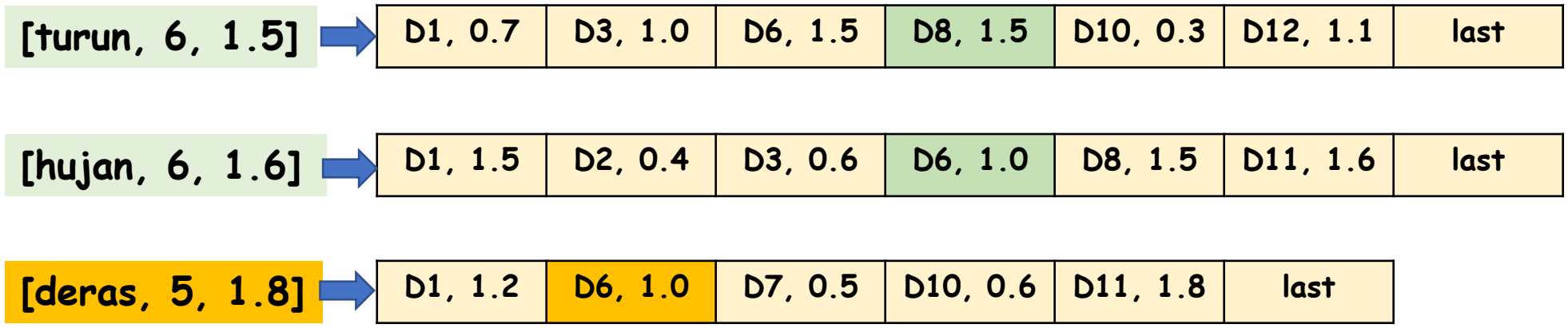
TOP-1 : [(D6,3.5)]
Threshold : 3.5
Pivot : 6
curDoc : 6

#documents that are fully evaluated: 2

```
1. Function next(θ)
2.    repeat
3.       /* Sort the terms in non decreasing order of
          DID */
4.       sort(terms, posting)
5.       /* Find pivot term - the first one with accumulated
          UB ≥ θ */
6.       pTerm ← findPivotTerm(terms, θ)
7.       if (pTerm = null) return (NoMoreDocs)
8.       pivot ← posting[pTerm].DID
9.       if (pivot = lastID) return (NoMoreDocs)
10.      if (pivot ≤ curDoc)
11.         /* pivot has already been considered, advance
             one of the preceding terms */
12.         aterm ← pickTerm(terms[0..pTerm])
13.         posting[aterm] ← aterm.iterator.next(curDoc+1)
14.      else /* pivot > curDoc */
15.         if (posting[0].DID = pivot)
16.            /* Success, all terms preceding pTerm belong
                to the pivot */
17.            curDoc ← pivot
18.            return (curDoc, posting)
19.         else
20.            /* not enough mass yet on pivot, advance
                one of the preceding terms */
21.            aterm ← pickTerm(terms[0..pTerm])
22.            posting[aterm] ← aterm.iterator.next(pivot)
23.      end repeat
```

Pointer dari term yang terpilih dimajukan hingga bertemu DID pertama yang >= 6 + 1 = 7

Yaitu D8



TOP-1 : [(D6,3.5)]
Threshold : 3.5
Pivot : 8
curDoc : 6

#documents that are fully evaluated: 2

```
1. Function next(θ)
2.   repeat
3.     /* Sort the terms in non decreasing order of
         DID */
4.     sort(terms, posting)
5.     /* Find pivot term - the first one with accumulated
         UB ≥ θ */
6.     pTerm ← findPivotTerm(terms, θ)
7.     if (pTerm = null) return (NoMoreDocs)
8.     pivot ← posting[pTerm].DID
9.     if (pivot = lastID) return (NoMoreDocs)
10.    if (pivot ≤ curDoc)
11.      /* pivot has already been considered, advance
           one of the preceding terms */
12.      aterm ← pickTerm(terms[0..pTerm])
13.      posting[aterm] ← aterm.iterator.next(curDoc+1)
14.    else /* pivot > curDoc */
15.      if (posting[0].DID = pivot)
16.        /* Success, all terms preceding pTerm belong
             to the pivot */
17.        curDoc ← pivot
18.        return (curDoc, posting)
19.      else
20.        /* not enough mass yet on pivot, advance
             one of the preceding terms */
21.        aterm ← pickTerm(terms[0..pTerm])
22.        posting[aterm] ← aterm.iterator.next(pivot)
23.    end repeat
```

[hujan, 6, 1.6] →	D1, 1.5	D2, 0.4	D3, 0.6	D6, 1.0	D8, 1.5	D11, 1.6	last
[deras, 5, 1.8] →	D1, 1.2	D6, 1.0	D7, 0.5	D10, 0.6	D11, 1.8	last	
[turun, 6, 1.5] →	D1, 0.7	D3, 1.0	D6, 1.5	D8, 1.5	D10, 0.3	D12, 1.1	last

TOP-1 : [(D6,3.5)]
Threshold : 3.5
Pivot : 8
curDoc : 6

#documents that are fully evaluated: 2

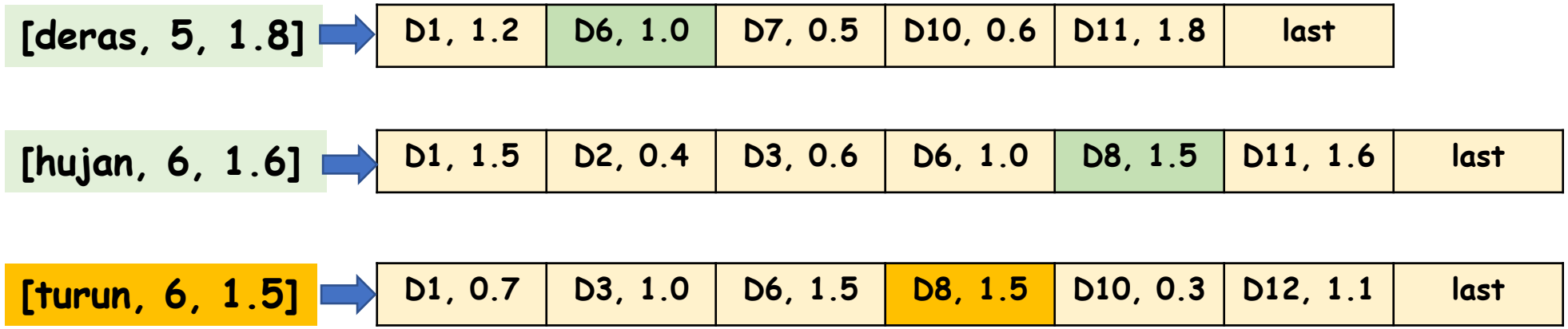
```
1. Function next(θ)
2.    repeat
3.       /* Sort the terms in non decreasing order of
          DID */
4.       sort(terms, posting)
5.       /* Find pivot term - the first one with accumulated
          UB ≥ θ */
6.       pTerm ← findPivotTerm(terms, θ)
7.       if (pTerm = null) return (NoMoreDocs)
8.       pivot ← posting[pTerm].DID
9.       if (pivot = lastID) return (NoMoreDocs)
10.      if (pivot ≤ curDoc)
11.         /* pivot has already been considered, advance
             one of the preceding terms */
12.         aterm ← pickTerm(terms[0..pTerm])
13.         posting[aterm] ← aterm.iterator.next(curDoc+1)
14.      else /* pivot > curDoc */
15.         if (posting[0].DID = pivot)
16.            /* Success, all terms preceding pTerm belong
                to the pivot */
17.            curDoc ← pivot
18.            return (curDoc, posting)
19.         else
20.            /* not enough mass yet on pivot, advance
                one of the preceding terms */
21.            aterm ← pickTerm(terms[0..pTerm])
22.            posting[aterm] ← aterm.iterator.next(pivot)
23.      end repeat
```

[hujan, 6, 1.6] →	D1, 1.5	D2, 0.4	D3, 0.6	D6, 1.0	D8, 1.5	D11, 1.6	last
[deras, 5, 1.8] →	D1, 1.2	D6, 1.0	D7, 0.5	D10, 0.6	D11, 1.8	last	
[turun, 6, 1.5] →	D1, 0.7	D3, 1.0	D6, 1.5	D8, 1.5	D10, 0.3	D12, 1.1	last

TOP-1 : [(D6,3.5)]
Threshold : 3.5
Pivot : 8
curDoc : 6

#documents that are fully evaluated: 2

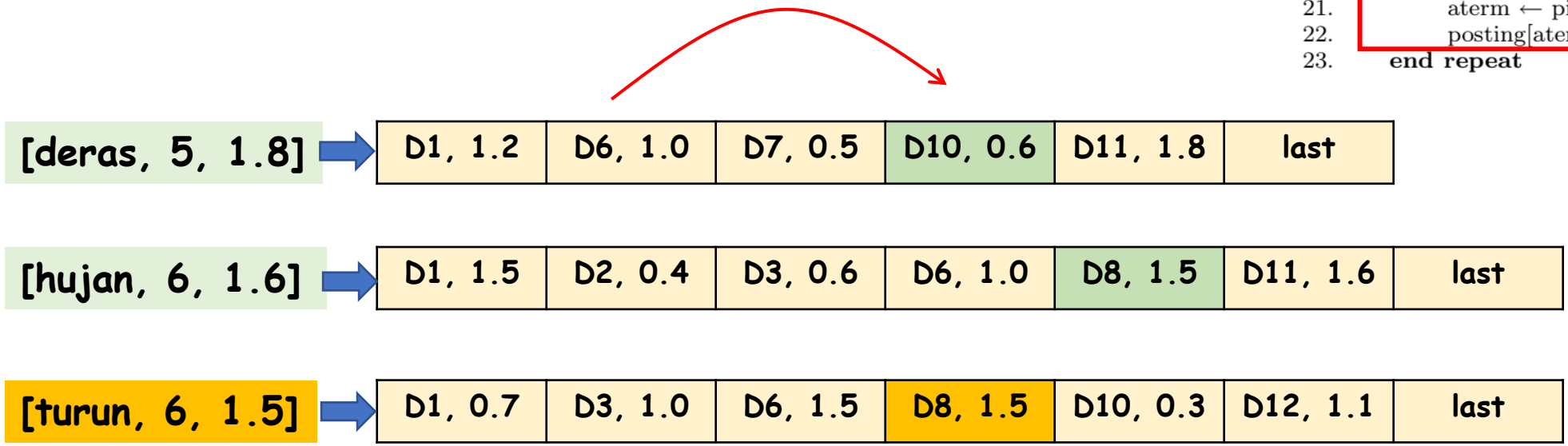
```
1. Function next(θ)
2.    repeat
3.       /* Sort the terms in non decreasing order of
          DID */
4.       sort(terms, posting)
5.       /* Find pivot term - the first one with accumulated
          UB ≥ θ */
6.       pTerm ← findPivotTerm(terms, θ)
7.       if (pTerm = null) return (NoMoreDocs)
8.       pivot ← posting[pTerm].DID
9.       if (pivot = lastID) return (NoMoreDocs)
10.      if (pivot ≤ curDoc)
11.         /* pivot has already been considered, advance
             one of the preceding terms */
12.         aterm ← pickTerm(terms[0..pTerm])
13.         posting[aterm] ← aterm.iterator.next(curDoc+1)
14.      else /* pivot > curDoc */
15.         if (posting[0].DID = pivot)
16.            /* Success, all terms preceding pTerm belong
                to the pivot */
17.            curDoc ← pivot
18.            return (curDoc, posting)
19.         else
20.            /* not enough mass yet on pivot, advance
                one of the preceding terms */
21.            aterm ← pickTerm(terms[0..pTerm])
22.            posting[aterm] ← aterm.iterator.next(pivot)
23.      end repeat
```



TOP-1 : [(D6,3.5)]
Threshold : 3.5
Pivot : 8
curDoc : 6

#documents that are fully evaluated: 2

```
1. Function next(θ)
2.    repeat
3.       /* Sort the terms in non decreasing order of
          DID */
4.       sort(terms, posting)
5.       /* Find pivot term - the first one with accumulated
          UB ≥ θ */
6.       pTerm ← findPivotTerm(terms, θ)
7.       if (pTerm = null) return (NoMoreDocs)
8.       pivot ← posting[pTerm].DID
9.       if (pivot = lastID) return (NoMoreDocs)
10.      if (pivot ≤ curDoc)
11.         /* pivot has already been considered, advance
             one of the preceding terms */
12.         aterm ← pickTerm(terms[0..pTerm])
13.         posting[aterm] ← aterm.iterator.next(curDoc+1)
14.      else /* pivot > curDoc */
15.         if (posting[0].DID = pivot)
16.            /* Success, all terms preceding pTerm belong
                to the pivot */
17.            curDoc ← pivot
18.            return (curDoc, posting)
19.         else
20.            /* not enough mass yet on pivot, advance
                one of the preceding terms */
21.            aterm ← pickTerm(terms[0..pTerm])
22.            posting[aterm] ← aterm.iterator.next(pivot)
23.      end repeat
```



TOP-1 : [(D6,3.5)]
Threshold : 3.5
Pivot : 10
curDoc : 6

#documents that are fully evaluated: 2

```
1. Function next(θ)
2.    repeat
3.       /* Sort the terms in non decreasing order of
          DID */
4.       sort(terms, posting)
5.       /* Find pivot term - the first one with accumulated
          UB ≥ θ */
6.       pTerm ← findPivotTerm(terms, θ)
7.       if (pTerm = null) return (NoMoreDocs)
8.       pivot ← posting[pTerm].DID
9.       if (pivot = lastID) return (NoMoreDocs)
10.      if (pivot ≤ curDoc)
11.         /* pivot has already been considered, advance
             one of the preceding terms */
12.         aterm ← pickTerm(terms[0..pTerm])
13.         posting[aterm] ← aterm.iterator.next(curDoc+1)
14.      else /* pivot > curDoc */
15.         if (posting[0].DID = pivot)
16.            /* Success, all terms preceding pTerm belong
                to the pivot */
17.            curDoc ← pivot
18.            return (curDoc, posting)
19.         else
20.            /* not enough mass yet on pivot, advance
                one of the preceding terms */
21.            aterm ← pickTerm(terms[0..pTerm])
22.            posting[aterm] ← aterm.iterator.next(pivot)
23.      end repeat
```

[hujan, 6, 1.6] →	D1, 1.5	D2, 0.4	D3, 0.6	D6, 1.0	D8, 1.5	D11, 1.6	last
[turun, 6, 1.5] →	D1, 0.7	D3, 1.0	D6, 1.5	D8, 1.5	D10, 0.3	D12, 1.1	last
[deras, 5, 1.8] →	D1, 1.2	D6, 1.0	D7, 0.5	D10, 0.6	D11, 1.8	last	

TOP-1 : [(D6,3.5)]
Threshold : 3.5
Pivot : 10
curDoc : 6

#documents that are fully evaluated: 2

```
1. Function next(θ)
2.    repeat
3.       /* Sort the terms in non decreasing order of
          DID */
4.       sort(terms, posting)
5.       /* Find pivot term - the first one with accumulated
          UB ≥ θ */
6.       pTerm ← findPivotTerm(terms, θ)
7.       if (pTerm = null) return (NoMoreDocs)
8.       pivot ← posting[pTerm].DID
9.       if (pivot = lastID) return (NoMoreDocs)
10.      if (pivot ≤ curDoc)
11.         /* pivot has already been considered, advance
             one of the preceding terms */
12.         aterm ← pickTerm(terms[0..pTerm])
13.         posting[aterm] ← aterm.iterator.next(curDoc+1)
14.      else /* pivot > curDoc */
15.         if (posting[0].DID = pivot)
16.            /* Success, all terms preceding pTerm belong
                to the pivot */
17.            curDoc ← pivot
18.            return (curDoc, posting)
19.         else
20.            /* not enough mass yet on pivot, advance
                one of the preceding terms */
21.            aterm ← pickTerm(terms[0..pTerm])
22.            posting[aterm] ← aterm.iterator.next(pivot)
23.   end repeat
```

[hujan, 6, 1.6] →	D1, 1.5	D2, 0.4	D3, 0.6	D6, 1.0	D8, 1.5	D11, 1.6	last
[turun, 6, 1.5] →	D1, 0.7	D3, 1.0	D6, 1.5	D8, 1.5	D10, 0.3	D12, 1.1	last
[deras, 5, 1.8] →	D1, 1.2	D6, 1.0	D7, 0.5	D10, 0.6	D11, 1.8	last	

TOP-1 : [(D6,3.5)]
Threshold : 3.5
Pivot : 11
curDoc : 6

#documents that are fully evaluated: 2

```
1. Function next(θ)
2.    repeat
3.       /* Sort the terms in non decreasing order of
          DID */
4.       sort(terms, posting)
5.       /* Find pivot term - the first one with accumulated
          UB ≥ θ */
6.       pTerm ← findPivotTerm(terms, θ)
7.       if (pTerm = null) return (NoMoreDocs)
8.       pivot ← posting[pTerm].DID
9.       if (pivot = lastID) return (NoMoreDocs)
10.      if (pivot ≤ curDoc)
11.         /* pivot has already been considered, advance
             one of the preceding terms */
12.         aterm ← pickTerm(terms[0..pTerm])
13.         posting[aterm] ← aterm.iterator.next(curDoc+1)
14.      else /* pivot > curDoc */
15.         if (posting[0].DID = pivot)
16.            /* Success, all terms preceding pTerm belong
                to the pivot */
17.            curDoc ← pivot
18.            return (curDoc, posting)
19.         else
20.            /* not enough mass yet on pivot, advance
                one of the preceding terms */
21.            aterm ← pickTerm(terms[0..pTerm])
22.            posting[aterm] ← aterm.iterator.next(pivot)
23.      end repeat
```

[turun, 6, 1.5] →	D1, 0.7	D3, 1.0	D6, 1.5	D8, 1.5	D10, 0.3	D12, 1.1	last
[deras, 5, 1.8] →	D1, 1.2	D6, 1.0	D7, 0.5	D10, 0.6	D11, 1.8	last	
[hujan, 6, 1.6] →	D1, 1.5	D2, 0.4	D3, 0.6	D6, 1.0	D8, 1.5	D11, 1.6	last

TOP-1 : [(D6,3.5)]
Threshold : 3.5
Pivot : 11
curDoc : 6

#documents that are fully evaluated: 2

```
1. Function next(θ)
2.    repeat
3.       /* Sort the terms in non decreasing order of
          DID */
4.       sort(terms, posting)
5.       /* Find pivot term - the first one with accumulated
          UB ≥ θ */
6.       pTerm ← findPivotTerm(terms, θ)
7.       if (pTerm = null) return (NoMoreDocs)
8.       pivot ← posting[pTerm].DID
9.       if (pivot = lastID) return (NoMoreDocs)
10.      if (pivot ≤ curDoc)
11.         /* pivot has already been considered, advance
             one of the preceding terms */
12.         aterm ← pickTerm(terms[0..pTerm])
13.         posting[aterm] ← aterm.iterator.next(curDoc+1)
14.      else /* pivot > curDoc */
15.         if (posting[0].DID = pivot)
16.            /* Success, all terms preceding pTerm belong
                to the pivot */
17.            curDoc ← pivot
18.            return (curDoc, posting)
19.         else
20.            /* not enough mass yet on pivot, advance
                one of the preceding terms */
21.            aterm ← pickTerm(terms[0..pTerm])
22.            posting[aterm] ← aterm.iterator.next(pivot)
23.   end repeat
```

[turun, 6, 1.5] →	D1, 0.7	D3, 1.0	D6, 1.5	D8, 1.5	D10, 0.3	D12, 1.1	last
[deras, 5, 1.8] →	D1, 1.2	D6, 1.0	D7, 0.5	D10, 0.6	D11, 1.8	last	
[hujan, 6, 1.6] →	D1, 1.5	D2, 0.4	D3, 0.6	D6, 1.0	D8, 1.5	D11, 1.6	last

TOP-1 : [(D6,3.5)]
Threshold : 3.5
Pivot : 12
curDoc : 6

#documents that are fully evaluated: 2

```
1. Function next(θ)
2.    repeat
3.       /* Sort the terms in non decreasing order of
          DID */
4.       sort(terms, posting)
5.       /* Find pivot term - the first one with accumulated
          UB ≥ θ */
6.       pTerm ← findPivotTerm(terms, θ)
7.       if (pTerm = null) return (NoMoreDocs)
8.       pivot ← posting[pTerm].DID
9.       if (pivot = lastID) return (NoMoreDocs)
10.      if (pivot ≤ curDoc)
11.         /* pivot has already been considered, advance
             one of the preceding terms */
12.         aterm ← pickTerm(terms[0..pTerm])
13.         posting[aterm] ← aterm.iterator.next(curDoc+1)
14.      else /* pivot > curDoc */
15.         if (posting[0].DID = pivot)
16.            /* Success, all terms preceding pTerm belong
                to the pivot */
17.            curDoc ← pivot
18.            return (curDoc, posting)
19.         else
20.            /* not enough mass yet on pivot, advance
                one of the preceding terms */
21.            aterm ← pickTerm(terms[0..pTerm])
22.            posting[aterm] ← aterm.iterator.next(pivot)
23.      end repeat
```

[deras, 5, 1.8] →	D1, 1.2	D6, 1.0	D7, 0.5	D10, 0.6	D11, 1.8	last
[hujan, 6, 1.6] →	D1, 1.5	D2, 0.4	D3, 0.6	D6, 1.0	D8, 1.5	D11, 1.6	last
[turun, 6, 1.5] →	D1, 0.7	D3, 1.0	D6, 1.5	D8, 1.5	D10, 0.3	D12, 1.1	last

TOP-1 : [(D6,3.5)]
Threshold : 3.5
Pivot : 12
curDoc : 6

#documents that are fully evaluated: 2

```
1. Function next(θ)
2.    repeat
3.       /* Sort the terms in non decreasing order of
          DID */
4.       sort(terms, posting)
5.       /* Find pivot term - the first one with accumulated
          UB ≥ θ */
6.       pTerm ← findPivotTerm(terms, θ)
7.       if (pTerm = null) return (NoMoreDocs)
8.       pivot ← posting[pTerm].DID
9.       if (pivot = lastID) return (NoMoreDocs)
10.      if (pivot ≤ curDoc)
11.         /* pivot has already been considered, advance
             one of the preceding terms */
12.         aterm ← pickTerm(terms[0..pTerm])
13.         posting[aterm] ← aterm.iterator.next(curDoc+1)
14.      else /* pivot > curDoc */
15.         if (posting[0].DID = pivot)
16.            /* Success, all terms preceding pTerm belong
                to the pivot */
17.            curDoc ← pivot
18.            return (curDoc, posting)
19.         else
20.            /* not enough mass yet on pivot, advance
                one of the preceding terms */
21.            aterm ← pickTerm(terms[0..pTerm])
22.            posting[aterm] ← aterm.iterator.next(pivot)
23.   end repeat
```

[deras, 5, 1.8] →	D1, 1.2	D6, 1.0	D7, 0.5	D10, 0.6	D11, 1.8	last	
[hujan, 6, 1.6] →	D1, 1.5	D2, 0.4	D3, 0.6	D6, 1.0	D8, 1.5	D11, 1.6	last
[turun, 6, 1.5] →	D1, 0.7	D3, 1.0	D6, 1.5	D8, 1.5	D10, 0.3	D12, 1.1	last

TOP-1 : [(D6,3.5)]
Threshold : 3.5
Pivot : last
curDoc : 6

#documents that are fully evaluated: 2

```
1. Function next(θ)
2.    repeat
3.       /* Sort the terms in non decreasing order of
          DID */
4.       sort(terms, posting)
5.       /* Find pivot term - the first one with accumulated
          UB ≥ θ */
6.       pTerm ← findPivotTerm(terms, θ)
7.       if (pTerm = null) return (NoMoreDocs)
8.       pivot ← posting[pTerm].DID
9.       if (pivot = lastID) return (NoMoreDocs)
10.      if (pivot ≤ curDoc)
11.         /* pivot has already been considered, advance
             one of the preceding terms */
12.         aterm ← pickTerm(terms[0..pTerm])
13.         posting[aterm] ← aterm.iterator.next(curDoc+1)
14.      else /* pivot > curDoc */
15.         if (posting[0].DID = pivot)
16.            /* Success, all terms preceding pTerm belong
                to the pivot */
17.            curDoc ← pivot
18.            return (curDoc, posting)
19.         else
20.            /* not enough mass yet on pivot, advance
                one of the preceding terms */
21.            aterm ← pickTerm(terms[0..pTerm])
22.            posting[aterm] ← aterm.iterator.next(pivot)
23.      end repeat
```

[hujan, 6, 1.6] →	D1, 1.5	D2, 0.4	D3, 0.6	D6, 1.0	D8, 1.5	D11, 1.6	last
[turun, 6, 1.5] →	D1, 0.7	D3, 1.0	D6, 1.5	D8, 1.5	D10, 0.3	D12, 1.1	last
[deras, 5, 1.8] →	D1, 1.2	D6, 1.0	D7, 0.5	D10, 0.6	D11, 1.8	last	

TOP-1 : [(D6,3.5)]
Threshold : 3.5
Pivot : last
curDoc : 6

#documents that are fully evaluated: 2

STOP !

```
1. Function next(θ)
2.   repeat
3.     /* Sort the terms in non decreasing order of
         DID */
4.     sort(terms, posting)
5.     /* Find pivot term - the first one with accumulated
         UB ≥ θ */
6.     pTerm ← findPivotTerm(terms, θ)
7.     if (pTerm = null) return (NoMoreDocs)
8.     pivot ← posting[pTerm].DID
9.     if (pivot = lastID) return (NoMoreDocs)
10.    if (pivot ≤ curDoc)
11.      /* pivot has already been considered, advance
           one of the preceding terms */
12.      aterm ← pickTerm(terms[0..pTerm])
13.      posting[aterm] ← aterm.iterator.next(curDoc+1)
14.    else /* pivot > curDoc */
15.      if (posting[0].DID = pivot)
16.        /* Success, all terms preceding pTerm belong
             to the pivot */
17.        curDoc ← pivot
18.        return (curDoc, posting)
19.      else
20.        /* not enough mass yet on pivot, advance
             one of the preceding terms */
21.        aterm ← pickTerm(terms[0..pTerm])
22.        posting[aterm] ← aterm.iterator.next(pivot)
23.    end repeat
```

[hujan, 6, 1.6] →	D1, 1.5	D2, 0.4	D3, 0.6	D6, 1.0	D8, 1.5	D11, 1.6	last
[turun, 6, 1.5] →	D1, 0.7	D3, 1.0	D6, 1.5	D8, 1.5	D10, 0.3	D12, 1.1	last
[deras, 5, 1.8] →	D1, 1.2	D6, 1.0	D7, 0.5	D10, 0.6	D11, 1.8	last	

TOP-2 :
Threshold :
Pivot :
curDoc :

Latihan

Lakukan hal yang sama dengan sebelumnya. Namun sekarang **Top-2** yang diambil!

[hujan, 6, 1.6] →	D1, 1.5	D2, 0.4	D3, 0.6	D6, 1.0	D8, 1.5	D11, 1.6	last
[turun, 6, 1.5] →	D1, 0.7	D3, 1.0	D6, 1.5	D8, 1.5	D10, 0.3	D12, 1.1	last
[deras, 5, 1.8] →	D1, 1.2	D6, 1.0	D7, 0.5	D10, 0.6	D11, 1.8	last	