Ανάκληση Πληροφορίας

Information Retrieval

Διδάσκων –
Δημήτριος Κατσαρός

Διάλεξη 7η

Κατασκευή του ευρετηρίου

- Πώς κατασκευάζουμε το ευρετήριο;
- Ποιες στρατηγικές μπορούμε ν'ακολουθήσουμε όταν έχουμε περιορισμό στην διαθέσιμη κύρια μνήμη;

- Πολλές από τις σχεδιαστικές αποφάσεις για ένα σύστημα IR βασίζονται στα χαρακτηριστικά του υλικού
- Θα κάνουμε μια επισκόπηση των βασικών εννοιών του υλικού που σχετίζονται με την σχεδίαση συστημάτων IR

Βασικές έννοιες του υλικού

- Η πρόσβαση στα δεδομένα της κύριας μνήμης είναι ***πολύ*** γρηγορότερη από ότι η πρόσβαση στα δεδομένα του μαγνητικού δίσκου
- Disk seeks: Τα δεδομένα δεν μπορούν να μεταφερθούν από τον δίσκο, όσο η κεφαλή του δίσκου τοποθετείται πάνω από αυτά
- Άρα: Η μεταφορά ενός μεγάλου τμήματος δεδομένων από τον δίσκο στην μνήμη είναι γρηγορότερη από ότι η μεταφορά πολλών μικρών
- Η I/O στον δίσκο γίνεται σε blocks: Ανάγνωση και εγγραφή ολόκληρων blocks (και όχι μικρών chunks)
- Μεγέθη block (οι επονομαζόμενες "*σελίδες*"): 8KB μέχρι 256 KB

Βασικές έννοιες του υλικού

- Οι εξυπηρετητές που χρησιμοποιούνται σε συστήματα IR έχουν πλέον αρκετά GB κύριας μνήμης
- Ο διαθέσιμος χώρος στον δίσκο είναι μερικές τάξεις (2–3) μεγέθους μεγαλύτερος
- Η ανοχή σε σφάλματα είναι πολύ ακριβή: Είναι πολύ πιο φθηνό να χρησιμοποιηθούν πολλές "κανονικές" μηχανές, παρά μια εξαιρετικής ευρωστίας, αλλά ακριβή μηχανή
  - Αυτό το ζήτημα αφορά στο distributed indexing με MapReduce, και δεν θα μας απασχολήσει στις διαλέξεις.
  - Ο ενδιαφερόμενος αναγνώστης παραπέμπεται στο μάθημα Advanced Distributed Computing http://inf-server.inf.uth.gr/courses/CE623/

Υποθέσεις για το υλικό

symbol	statistic	value
s	average seek time	5 ms = 5 x 10^{-3} s
b	transfer time per byte	0.02 μs = 2 x 10^{-8} s
	processor's clock rate	10^{9} s^{-1}
p	low-level operation (e.g., compare & swap a word)	0.01 μs = 10^{-8} s
	size of main memory	several GB
	size of disk space	1 TB or more

RCV1: Η συλλογή των εγγράφων

- Η συλλογή των έργων του Shakespeare δεν είναι αρκετά μεγάλη για να επιδειχτούν πολλά από τα θέματα που μας αφορούν στο συγκεκριμένο μάθημα
- Η συλλογή στην οποία θ' αναφερθούμε δεν είναι πάρα πολύ μεγάλη, αλλά είναι διαθέσιμη δημοσίως και είναι σαφώς μεγαλύτερη
- Ως παράδειγμα εφαρμογής κλιμακούμενων αλγορίθμων κατασκευής του ευρετηρίου, θα χρησιμοποιήσουμε την συλλογή Reuters RCV1
- Περιλαμβάνει ένα έτος ανταποκρίσεων του Reuters (μέρος του 1995 και 1996)

Ένα έγγραφο της συλλογής Reuters RCV1

REUTERS

You are here: Home > News > Science > Article

Go to a Section: U.S. International Business Markets Politics Entertainment Technology Sports Oddly Enoug

Extreme conditions create rare Antarctic clouds

Tue Aug 1, 2006 3:20am ET

Email This Article | Print This Article | Reprints

[-] Text [+]

SYDNEY (Reuters) - Rare, mother-of-pearl colored clouds caused by extreme weather conditions above Antarctica are a possible indication of global warming, Australian scientists said on Tuesday.

Known as nacreous clouds, the spectacular formations showing delicate wisps of colors were photographed in the sky over an Australian meteorological base at Mawson Station on July 25.

Στατιστικά της συλλογής Reuters RCV1

symbol	statistic	value
N	documents	800,000
L	avg. # tokens per doc	200
M	terms (= word types)	400,000
	avg. # bytes per token (incl. spaces/punct.)	6
	avg. # bytes per token (without spaces/punct.)	4.5
	avg. # bytes per term	7.5
	non-positional postings	100,000,000

4.5 bytes per word token versus 7.5 bytes per word type: Γιατί;

Θυμηθείτε την βασική ιδέα κατασκευής

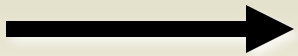
- Τα έγγραφα γίνονται parse για να εξαχθούν οι λέξεις, και αυτές κατόπιν αποθηκεύονται με το Document ID

Doc 1

I did enact Julius Caesar I was killed i' the Capitol; Brutus killed me

Doc 2

So let it be with Caesar. The noble Brutus hath told you Caesar was ambitious



Term	Doc #
I	1
did	1
enact	1
julius	1
caesar	1
I	1
was	1
killed	1
i'	1
the	1
capitol	1
brutus	1
killed	1
me	1
so	2
let	2
it	2
be	2
with	2
caesar	2
the	2
noble	2
brutus	2
hath	2
told	2
you	2
caesar	2
was	2
ambitious	2

Το βασικό βήμα

- Όταν όλα τα έγγραφα έχουν γίνει parse, το inverted file <u>ταξινομείται</u> ως προς τους όρους

Εστιάζουμε σ' αυτό το βήμα ταξινόμησης.
Πρέπει να ταξινομήσουμε 100M items.

Term	Doc #
I	1
did	1
enact	1
julius	1
caesar	1
I	1
was	1
killed	1
i'	1
the	1
capitol	1
brutus	1
killed	1
me	1
so	2
let	2
it	2
be	2
with	2
caesar	2
the	2
noble	2
brutus	2
hath	2
told	2
you	2
caesar	2
was	2
ambitious	2

Term	Doc #
ambitious	2
be	2
brutus	1
brutus	2
capitol	1
caesar	1
caesar	2
caesar	2
did	1
enact	1
hath	1
I	1
I	1
i'	1
it	2
julius	1
killed	1
killed	1
let	2
me	1
noble	2
so	2
the	1
the	2
told	2
you	2
was	1
was	2
with	2

Κλιμακώνοντας την κατασκευή του index

- Η κατασκευή εντός της κύριας μνήμης δεν κλιμακώνεται ως προς τον αριθμό των postings (απλά, δεν χωράνε όλοι στην κύρια μνήμη)
- Πώς μπορούμε να κατασκευάσουμε το ευρετήριο για εξαιρετικά μεγάλες συλλογές εγγράφων;
- Λαμβάνοντας υπόψη τους περιορισμούς που επιβάλλει το υλικό
  - Μνήμη, δίσκος, ταχύτητα, κ.τ.λ.

Κατασκευή βασισμένη σε ταξινόμηση

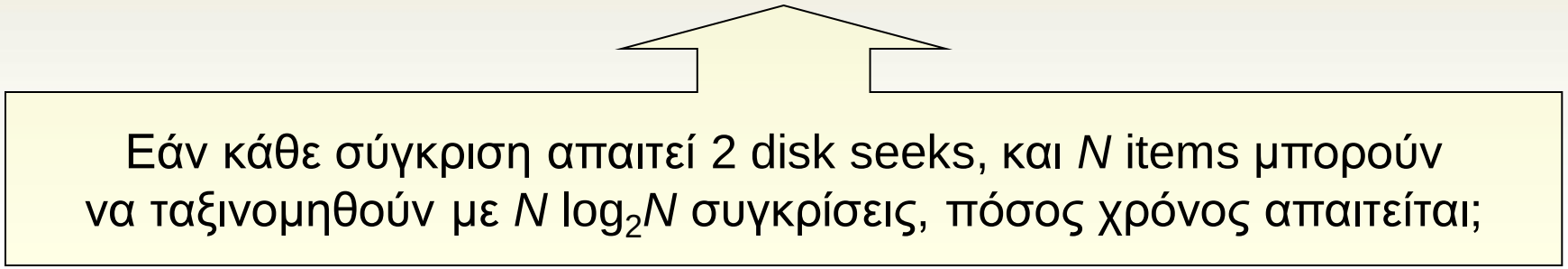
- Καθώς κατασκευάζουμε τον index, κάνουμε parse τα έγγραφα, ένα κάθε φορά
  - Κατά την κατασκευή, δεν μπορούμε εύκολα να εκμεταλλευτούμε τεχνικές συμπίεσης
- Τα τελικά postings για κάθε όρο είναι ημιτελή μέχρι να τελειώσουμε την κατασκευή
- Με 12 bytes per non-positional postings entry *(term, doc, freq)*, απαιτείται πάρα πολύς χώρος για μεγάλες συλλογές
- T = 100,000,000 στην περίπτωση του RCV1
  - Οι Web συλλογές είναι τεράστιες: π.χ., οι *New York Times* παρέχουν index για πάνω από 150 έτη άρθρων
- Επομένως: Πρέπει ν' αποθηκεύσουμε ενδιάμεσα αποτελέσματα στον δίσκο

Χρήση του ίδιου αλγορίθμου για δίσκο;

- Μπορούμε να χρησιμοποιήσουμε τον ίδιο αλγόριθμο κατασκευής για τεράστιες συλλογές, αλλά κάνοντας χρήση του δίσκου αντί της κύριας μνήμης;
- **ΌΧΙ**: Η ταξινόμηση Τ = 100,000,000 εγγραφών στον δίσκο είναι εξαιρετικά αργή – πάρα πολλά disk seeks
- Χρειαζόμαστε έναν αλγόριθμο για *external sorting*

Στενωπός

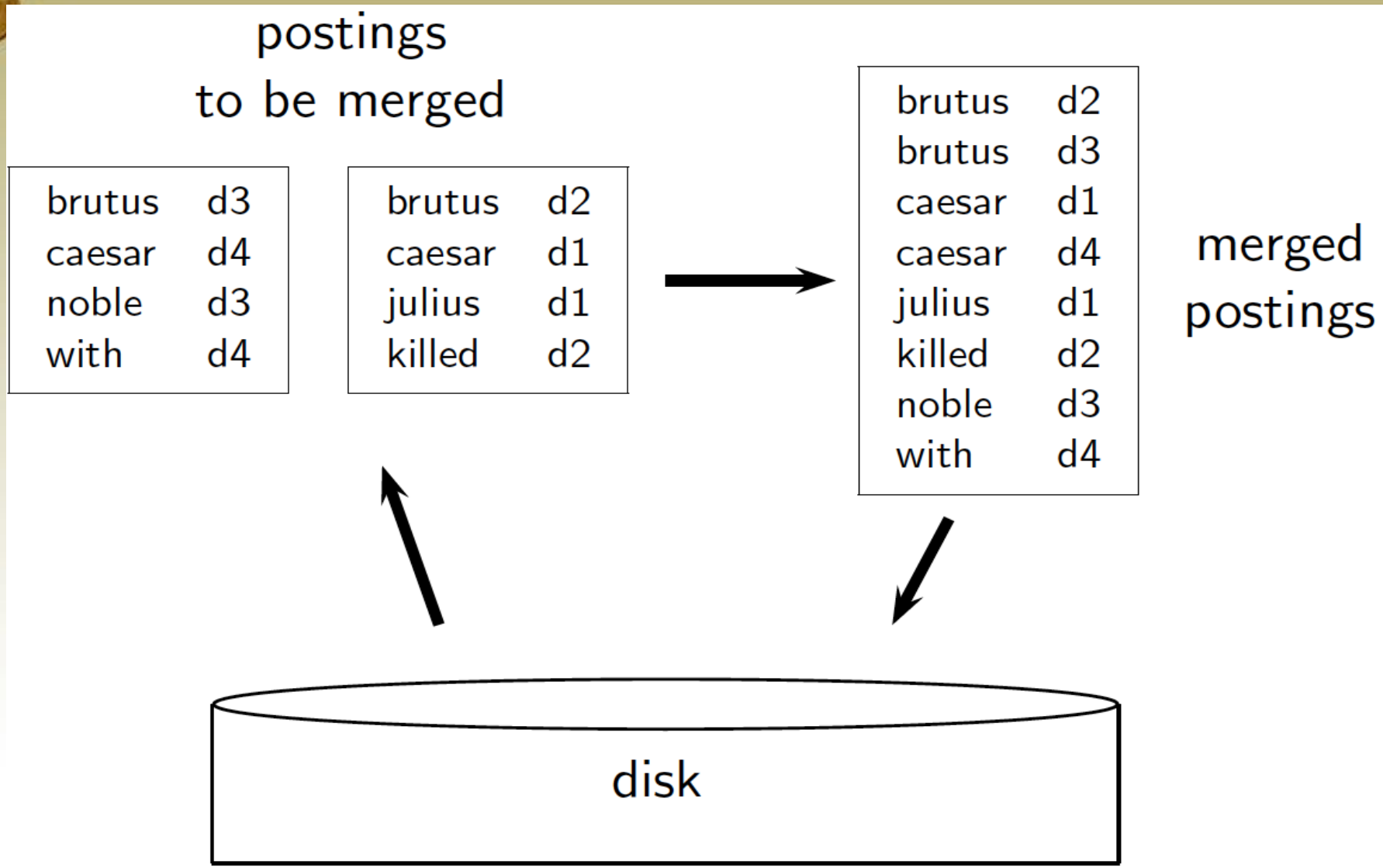
- Κάνουμε parse και χτίζουμε postings entries, ένα έγγραφο κάθε φορά
- Ταξινομούμε τις postings entries ανά όρο (κατόπιν, ανά έγγραφο εντός κάθε όρου)
- Κάνοντας αυτό με τυχαία disk seeks θα ήταν εξαιρετικά αργό – πρέπει να ταξινομηθούν T=100M εγγραφές



BSBI: Blocked Sort-Based Indexing (Ταξινόμηση με λιγότερα disk seeks)

- 12-byte (4+4+4) εγγραφές: *(term, doc, freq)*
- Παράγονται καθώς κάνουμε parse τα έγγραφα
- Πρέπει να ταξινομηθούν 100M τέτοιες 12-byte εγγραφές ως προς τους *όρους*
- Ορίζουμε ένα Block ~ 10M τέτοιων εγγραφών
  - Μπορούν εύκολα να χωρέσουν δύο τέτοια στην μνήμη
  - Θα έχουμε 10 τέτοια blocks για να δουλέψουμε
- Βασικη ιδέα του αλγορίθμου:
  - Συσσωρεύουμε postings από κάθε block, τις ταξινομούμε, και τις γράφουμε στον δίσκο
  - Δηλαδή, συγχωνεύουμε τα blocks σε ένα μεγαλύτερο, ταξινομημένο

Επίδειξη της βασικής ιδέας



Ταξινόμηση 10 blocks των 10M records

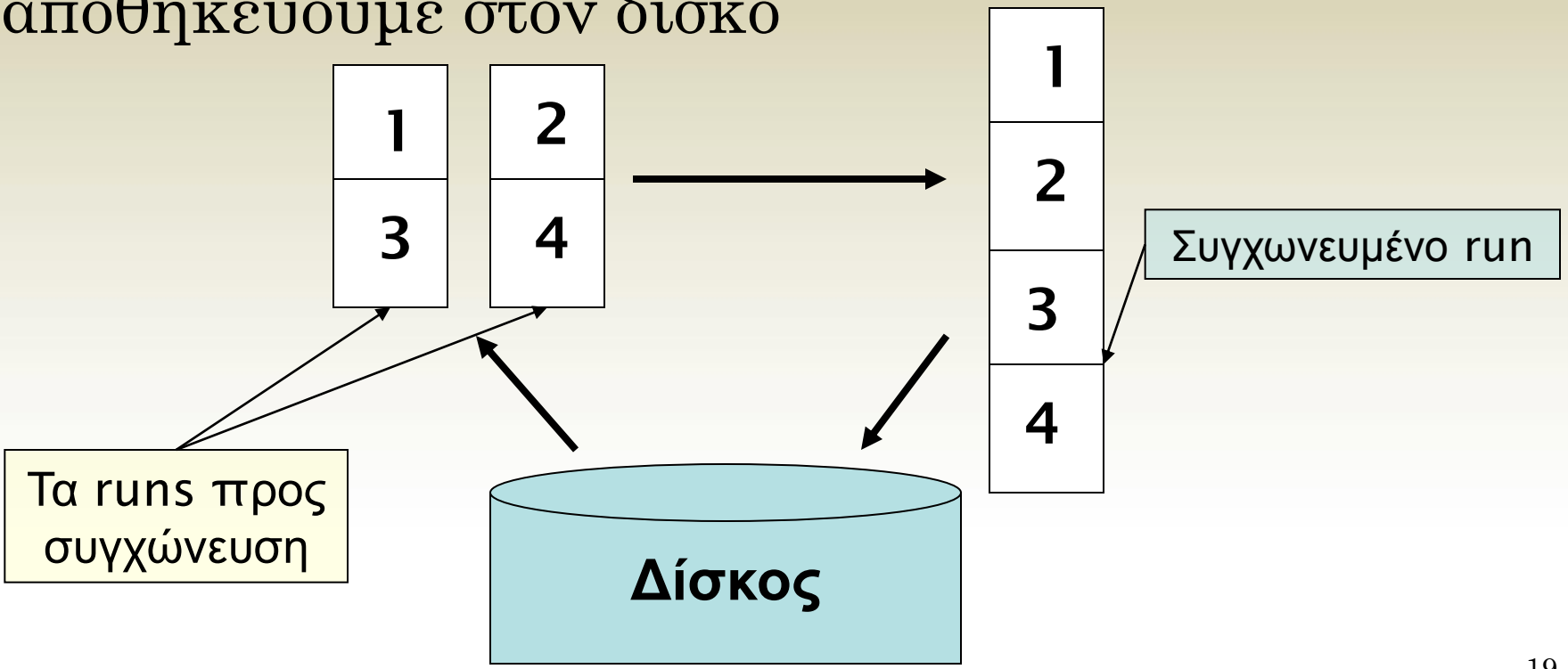
- Πρώτα, διαβάζουμε κάθε block και ταξινομούμε:
  - Ο quicksort απαιτεί $2N \ln N$ αναμενόμενα βήματα
  - Στην περίπτωσή μας: 2 x (10M ln 10M) βήματα
- 10 φορές αυτήν την εκτίμηση – μας δίνει 10 sorted *runs* των 10M εγγραφών η κάθε μια
- Γίνεται εύκολα, απαιτεί 2 αντίγραφα των δεδομένων στον δίσκο
  - Αλλά, μπορούμε να το βελτιστοποιήσουμε αυτό
    - Δείτε το Πρόβλημα-05 στην Σειρά Προβλημάτων $2^η$

Ο αλγόριθμος BSBI

```
BSBIndexConstruction()
 1  n ← 0
 2  while (all documents have not been processed)
 3  do n ← n + 1
 4     block ← ParseNextBlock()
 5     BSBI-Invert(block)
 6     WriteBlockToDisk(block, f_n)
 7  MergeBlocks(f_1, ..., f_n; f_merged)
```

Πώς να συγχωνεύσουμε τα ταξινομημένα runs;

- Μπορούμε να εκτελέσουμε binary merges, με ένα merge tree των $\log_2 10 = 4$ στρωμάτων
- Σε κάθε στρώμα, φέρνουμε στην μνήμη τα runs σε blocks των 10M, εκτελούμε συγχώνευση, και τα αποθηκεύουμε στον δίσκο



Πώς να συγχωνεύσουμε τα ταξινομημένα runs;

- Αλλά είναι πιο αποδοτικό να εκτελέσουμε ένα *n*-way merge, όπου διαβάζουμε από όλα τα blocks ταυτόχρονα
- Υπό την προϋπόθεση ότι φέρνουμε "κατάλληλου" μεγέθους chunks από κάθε block στην μνήμη, και κατόπιν γράφουμε στον δίσκο ένα "κατάλληλου" μεγέθους chunk στην έξοδο, τότε δεν κινδυνεύουμε από πολλά disk seeks

Παραμένον πρόβλημα με τον sort-based αλγόριθμο

- Μια βασική υπόθεση που κάναμε πριν: μπορούμε να αποθηκεύσουμε το dictionary στην μνήμη
- Χρειαζόμαστε το dictionary (που μεγαλώνει δυναμικά) για να κάνουμε την αντιστοίχιση term προς termID
- Στην πραγματικότητα, μπορούμε να εργαστούμε με term,docID postings αντί για termID,docID postings . . .
- . . . αλλά τότε τα ενδιάμεσα αρχεία γίνονται εξαιρετικά μεγάλα. (Θα είχαμε έτσι μια scalable, αλλά πολύ αργή μεθοδο κατασκευής του index

SPIMI: Single-Pass In-Memory Indexing

- Βασική ιδέα 1: Γεννούμε ξεχωριστά dictionaries για κάθε block – δεν χρειάζεται να διατηρούμε αντιστοίχιση *term-termID* μεταξύ των blocks
- Βασική ιδέα 2: Δεν ταξινομούμε. Συσσωρεύουμε postings στις postings lists καθώς αυτά γεννιούνται
- Με αυτές τις δυο ιδέες, μπορούμε να κατασκευάσουμε έναν πλήρη inverted index για κάθε block
- Αυτοί οι ξεχωριστοί indexes μπορούν κατόπιν να συγχωνευτούν σε έναν "μεγάλο" index

Ο αλγόριθμος SPIMI

```
SPIMI-INVERT(token_stream)
 1  output_file = NEWFILE()
 2  dictionary = NEWHASH()
 3  while (free memory available)
 4  do token ← next(token_stream)
 5     if term(token) ∉ dictionary
 6        then postings_list = ADDTODICTIONARY(dictionary, term(token))
 7        else postings_list = GETPOSTINGSLIST(dictionary, term(token))
 8     if full(postings_list)
 9        then postings_list = DOUBLEPOSTINGSLIST(dictionary, term(token))
10     ADDTOPOSTINGSLIST(postings_list, docID(token))
11  sorted_terms ← SORTTERMS(dictionary)
12  WRITEBLOCKTODISK(sorted_terms, dictionary, output_file)
13  return output_file
```

- Η συγχώνευση των blocks είναι ανάλογη εκείνης του BSBI

SPIMI: Συμπίεση

- Η συμπίεση καθιστά τον αλγόριθμο SPIMI ακόμη περισσότερο αποδοτικό
  - Συμπίεση των όρων
  - Συμπίεση των postings
- Δείτε την επόμενη διάλεξη