

ΜΥΕ003: Ανάκτηση Πληροφορίας

Διδάσκουσα: Ευαγγελία Πιτουρά

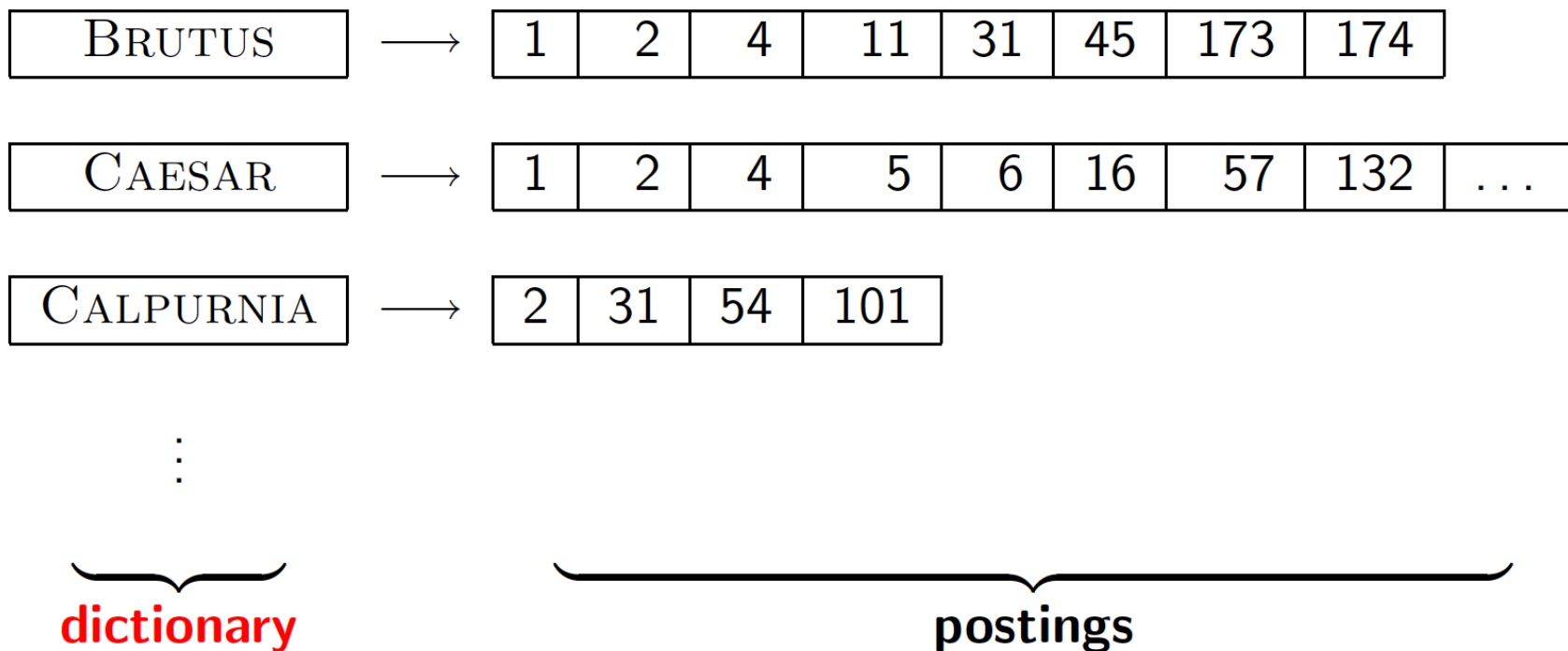
Κεφάλαια 4, 5: Κατασκευή Ευρετηρίου. Στατιστικά Συλλογής.
Συμπύεση

Τι θα δούμε σήμερα

- Κατασκευή ευρετηρίου
- Στατιστικά για τη συλλογή
- Συμπύεση

ΚΑΤΑΣΚΕΥΗ ΕΥΡΕΤΗΡΙΟΥ

Η βασική δομή: Το αντεστραμμένο ευρετήριο (inverted index)



Λεξικό: οι *όροι* (term) και η συχνότητα εγγράφων (#εγγράφων της συλλογής που εμφανίζονται)

Λίστες καταχωρήσεων (posting lists)
Κάθε καταχώρηση (posting) για ένα όρο περιέχει μια *διατεταγμένη λίστα με τα έγγραφα* (DocID) στα οποία εμφανίζεται ο όρος – συχνά επιπρόσθετα στοιχεία, όπως position, term frequency, κλπ

Κατασκευή ευρετηρίου

- Πως κατασκευάζουμε το ευρετήριο (indexing, indexers)
- Ποιες στρατηγικές χρησιμοποιούμε όταν έχουμε περιορισμένη μνήμη

Βασικά στοιχεία του υλικού

- Πολλές αποφάσεις στην ανάκτηση πληροφορίας βασίζονται στα *χαρακτηριστικά του διαθέσιμου υλικού* (διαφορετικοί αλγόριθμοι/τεχνικές)
- Ας δούμε μερικά βασικά χαρακτηριστικά

Βασικά χαρακτηριστικά του υλικού

- Η προσπέλαση δεδομένων στην κύρια μνήμη είναι πολύ *πιο γρήγορη* από την προσπέλαση δεδομένων στο δίσκο (περίπου ένας παράγοντας του 10) – *ιεραρχία μνήμης* (register-cache L1, L2, L3, RAM (main memory), hard disk)
 - Σημαντική τεχνική: *caching*
- Disk seeks (χρόνος αναζήτησης): Ενώ τοποθετείται η κεφαλή δε γίνεται μεταφορά δεδομένων
 - Άρα: Η *μεταφορά μεγάλων κομματιών* (chunk) δεδομένων από το δίσκο στη μνήμη είναι γρηγορότερη από τη μεταφορά πολλών μικρών
- Η επικοινωνία με το δίσκο (Disk I/O) γίνεται *σε σελίδες (block-based)*:
 - Διαβάζονται και γράφονται ολόκληρα blocks (όχι τμήματά τους). Σχετικός χώρος στη μνήμη – memory buffer, Μέγεθος Block: 8KB - 256KB. (*locality*)
- Παράλληλα με την επεξεργασία δεδομένων
 - *Συμπίεση* – συχνά πιο γρήγορη *αποσυμπίεση + επεξεργασία*

Βασικά χαρακτηριστικά του υλικού

- Οι επεξεργαστές που χρησιμοποιούνται στην ΑΠ διαθέτουν πολλά *GB κύριας μνήμης*, συχνά δεκάδες από GBs.
- Ο διαθέσιμος χώρος δίσκου είναι πολλές (2–3) τάξεις μεγαλύτερος.
- Η ανοχή στα σφάλματα (fault tolerance) είναι πολύ ακριβή: φθηνότερο να χρησιμοποιεί κανείς πολλές κανονικές μηχανές παρά μια «μεγάλη»

Υποθέσεις για το υλικό (~2008)

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

Η συλλογή RCV1

- Η συλλογή με τα άπαντα του Shakespeare δεν είναι αρκετά μεγάλη για το σκοπό της σημερινής διάλεξης.
- Η συλλογή που θα χρησιμοποιήσουμε δεν είναι στην πραγματικότητα πολύ μεγάλη, αλλά είναι διαθέσιμη στο κοινό.
- Θα χρησιμοποιήσουμε τη συλλογή **RCV1**.
 - Είναι ένας χρόνος του κυκλώματος ειδήσεων του Reuters (Reuters newswire) (μέρος του 1995 και 1996)
 - *1GB κειμένου*

Ένα έγγραφο της συλλογής Reuters RCV1



You are here: [Home](#) > [News](#) > [Science](#) > [Article](#)

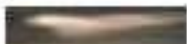
Go to a Section: [U.S.](#) [International](#) [Business](#) [Markets](#) [Politics](#) [Entertainment](#) [Technology](#) [Sports](#) [Oddly Enough](#)

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.

Η συλλογή RCV1: στατιστικά

N	documents	800,000
L	tokens per document	200
M	terms (= word types)	400,000
	bytes per token (incl. spaces/punct.)	6
	bytes per token (without spaces/punct.)	4.5
	bytes per term (= word type)	7.5
T	non-positional postings	100,000,000

- *Γιατί κατά μέσο ένα term είναι μεγαλύτερο από ένα token;*

Κατασκευή ευρετηρίου

1^ο πέρασμα:

Επεξεργάζομαστε τα έγγραφα για να βρούμε τις λέξεις, δημιουργούμε ζεύγη (term, doc-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

Βασικό βήμα: sort

2^ο πέρασμα: αφού έχουμε επεξεργαστεί όλα τα έγγραφα, *γίνεται διάταξη* (sort) με βάση τους όρους (και δευτερευόντως το doc-id) και κατασκευάζουμε το ευρετήριο

- Υπολογίζουμε και όποια επιπρόσθετη πληροφορία χρειάζεται (πχ frequency)
- Συμφέρει να χρησιμοποιούμε termID αντί term

Θα συζητήσουμε πως γίνεται η διάταξη αν δεν επαρκεί η μνήμη

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

Κλιμάκωση της κατασκευής του ευρετηρίου

- Για μεγάλες συλλογές, δεν είναι δυνατή η πλήρης κατασκευή του ευρετηρίου στη μνήμη (in-memory)
 - Δεν μπορούμε να φορτώσουμε όλη τη συλλογή στη μνήμη, να την ταξινομήσουμε και να γράψουμε το ευρετήριο πίσω στο δίσκο
- Πως μπορούμε να κατασκευάσουμε ένα ευρετήριο για μια πολύ μεγάλη συλλογή;
 - Μας ενδιαφέρει το I/O κόστος

External sorting

Κατασκευή με βάση τη διάταξη

Κάποιοι υπολογισμοί:

- Κάθε εγγραφή καταχώρησης (ακόμα και χωρίς πληροφορία θέσης - non-positional) δηλαδή (*termID*, *docID*, *freq*) καταλαμβάνει $4+4+4 = 12$ bytes και απαιτεί πολύ χώρο για μεγάλες συλλογές
 - Χρήση *termID* αντί *term* για καλύτερη απόδοση (400.000 όροι => 4 bytes για ids vs 7,5 για το *term*)
 - Χρήση *docID* (800.000 έγγραφα)
- $T = 100.000.000$ όροι για το RCV1 (**1.2 GB**)
 - Αυτή η συλλογή χωράει στη μνήμη, αλλά στην πραγματικότητα πολύ μεγαλύτερες, Π.χ., οι *New York Times* παρέχουν ένα ευρετήριο για κύκλωμα ειδήσεων >150 χρόνια
- Πρέπει να αποθηκεύουμε ενδιάμεσα αποτελέσματα στο δίσκο
- Καθώς κατασκευάζουμε το ευρετήριο, επεξεργαζόμαστε τα έγγραφα ένα-ένα
- Οι τελικές *καταχωρήσεις* για κάθε όρο είναι *ημιτελής* μέχρι το τέλος

Διάταξη χρησιμοποιώντας το δίσκο σαν «μνήμη»;

- Μπορούμε να χρησιμοποιήσουμε τον ίδιο αλγόριθμο διάταξης για την κατασκευή του ευρετηρίου αλλά χρησιμοποιώντας *δίσκο αντί για μνήμη*;
- Όχι: Διάταξη $T = 100,000,000$ εγγραφών στο δίσκο είναι πολύ αργή – πολλές τυχαίες ανακτήσεις (disk seeks)
 - Θα χρειαστούν μέρες

BSBI: Αλγόριθμος κατασκευής κατά block (Blocked sort-based Indexing)

1. Χώρισε τη συλλογή εγγράφων σε κομμάτια ίσου μεγέθους
2. Ταξινόμησε τα ζεύγη termID–docID για κάθε κομμάτι στη μνήμη
3. Αποθήκευσε τα ενδιάμεσα αποτελέσματα (**runs**) στο δίσκο
4. Συγχώνευσε τα ενδιάμεσα αποτελέσματα

Διάταξη 10 blocks των 10M εγγραφών

- Εγγραφές 12-byte (4+4+4) (*termID, docID, freq*).
- Παράγονται κατά τη διάσχιση των εγγράφων
- Διάταξη 100M τέτοιων 12-byte εγγραφών με βάση τον όρο.
- Ορίζουμε ένα **Block** ~ 10M τέτοιες εγγραφές
 - Μπορούμε εύκολα να έχουμε κάποια από αυτά στη μνήμη.
 - Αρχικά, 10 τέτοια blocks.

Αλγόριθμος BSBI

Βασική ιδέα:

- Συγκέντρωσε καταχωρήσεις για να γεμίσει ένα block, διάταξε τις καταχωρήσεις σε κάθε block, γράψε το στο δίσκο (**run**)
- Μετά συγχώνευσε τα blocks σε ένα μεγάλο διατεταγμένο block.

BSBINDEXCONSTRUCTION()

```

1   $n \leftarrow 0$ 
2  while (all documents have not been processed)
3  do  $n \leftarrow n + 1$ 
4      $block \leftarrow \text{PARSENEXTBLOCK}()$ 
5     BSBI-INVERT( $block$ )
6     WRITEBLOCKTODISK( $block, f_n$ )
7  MERGEBLOCKS( $f_1, \dots, f_n; f_{\text{merged}}$ )

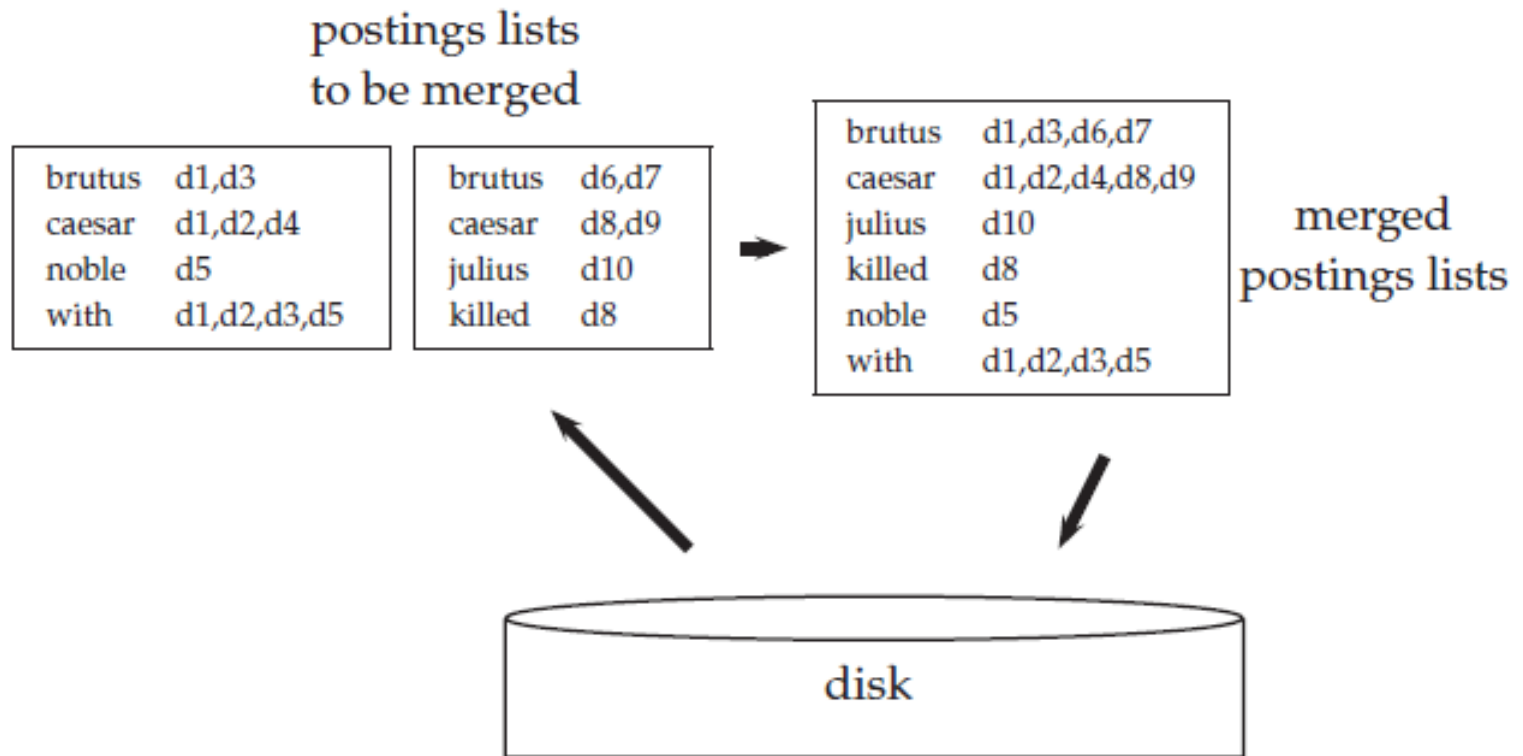
```

Διάβασε ένα-ένα τα έγγραφα – γεμίζοντας ένα block με <termid, docid>,

Invert: (1) διάταξη ζευγών με βάση το termid, (2) συγκέντρωση όλων με το ίδιο termid σε postings,

Γράψε το γεμάτο block στο δίσκο

Παράδειγμα



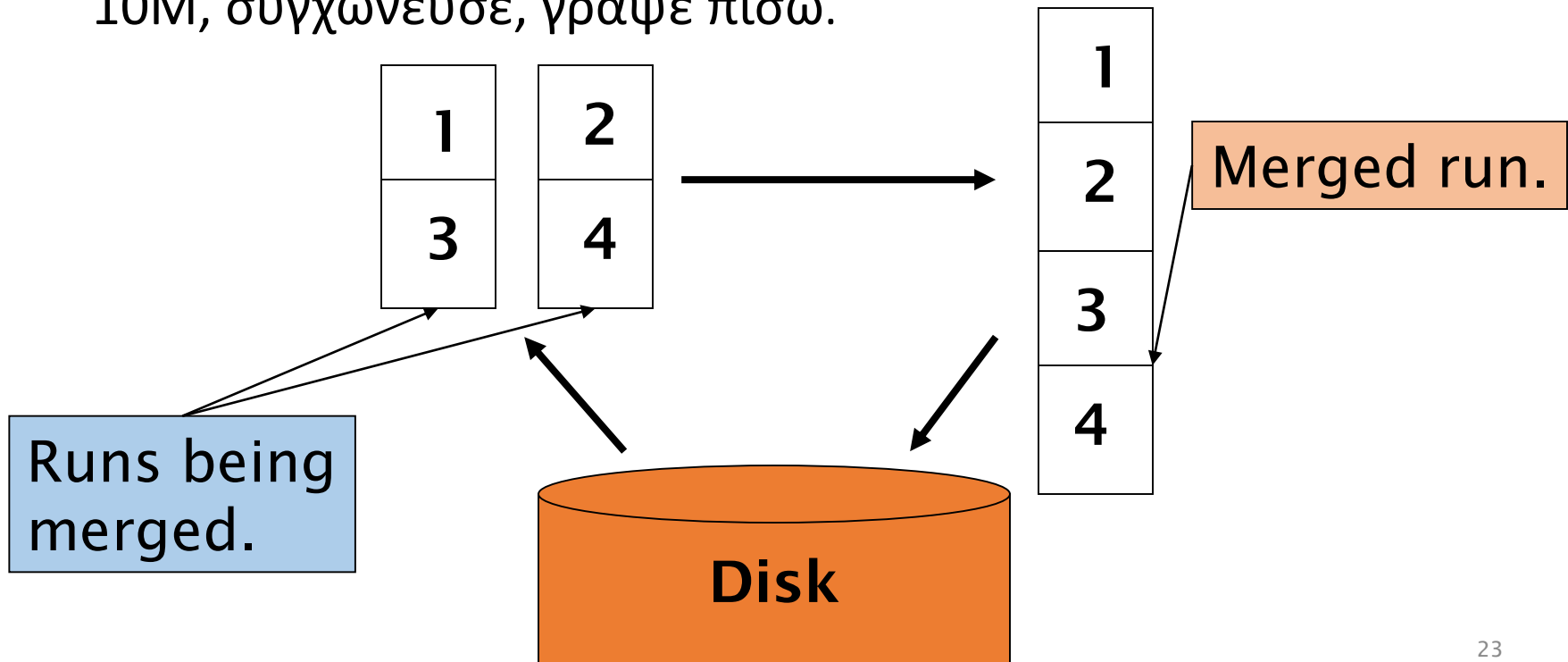
merge sort

Διάταξη 10 blocks των 10M εγγραφών

- Πρώτα, διάβασε κάθε block και διάταξε τις εγγραφές του:
 - Quicksort $2N \ln N$ αναμενόμενα βήματα
 - Στην περίπτωσή μας, $2 \times (10M \ln 10M)$ steps
- *Άσκηση: εκτιμήστε το συνολικό κόστος για να διαβάσουμε κάθε block από το δίσκο και να εφαρμόσουμε quicksort σε αυτό.*
- 10 φορές αυτή η εκτίμηση του χρόνου μας δίνει 10 διατεταγμένα runs των 10M εγγραφών το καθένα.
- Ο απλός τρόπος χρειάζεται 2 αντίγραφα των δεδομένων στο δίσκο
 - Αλλά μπορεί να βελτιωθεί

Πως θα γίνει η συγχώνευση των runs?

- Δυαδική συγχώνευση, μια δεντρική δομή με $\log_2 10 = 4$ επίπεδα.
- Σε κάθε επίπεδο, διάβασε στη μνήμη runs σε blocks των 10M, συγχώνευσε, γράψε πίσω.



Πως θα γίνει η συγχώνευση των runs?

- Πιο αποδοτικά με μια **multi-way συγχώνευση**, όπου διαβάζουμε από όλα τα blocks ταυτόχρονα
 - Κρατάμε ανοιχτά και τα 10
 - Μικροί read buffers και ένα write buffer
 - Σε κάθε επανάληψη, βρίσκουμε το μικρότερο termid που δεν έχουμε επεξεργαστεί, και διαβάζουμε όλες τις καταχωρήσεις για αυτό
- Υπό την προϋπόθεση ότι διαβάζουμε στη μνήμη αρκετά μεγάλα κομμάτια κάθε block και μετά γράφουμε πίσω αρκετά μεγάλα κομμάτια, αλλιώς πάλι πρόβλημα με τις αναζητήσεις στο δίσκο

BSBI: περίληψη

- Βασική ιδέα:
 - Διάβαζε τα έγγραφα, συγκέντρωσε $\langle \text{termid}, \text{docid} \rangle$ καταχωρήσεις έως να γεμίσει ένα block, διάταξε τις καταχωρήσεις σε κάθε block, γράψε το στο δίσκο.
 - Μετά συγχώνευσε τα blocks σε ένα μεγάλο διατεταγμένο block.
- Δυαδική συγχώνευση, μια δεντρική δομή με $\log_2 B$ επίπεδα, όπου B ο αριθμός των blocks.
- Multi-way merge

Χρήση αναγνωριστικού όρου (termID)

- Υπόθεση: *κρατάμε το λεξικό στη μνήμη*
- Χρειαζόμαστε το λεξικό (το οποίο μεγαλώνει δυναμικά) για να υλοποιήσουμε την απεικόνιση μεταξύ όρου (term) σε termID (όταν ένας όρος, κοιτάμε αν ήδη map σε ID, κλπ)
- Θα μπορούσαμε να εργαστούμε και με term, docID καταχωρήσεις αντί των termID, docID καταχωρήσεων, αλλά τα ενδιάμεσα αρχεία γίνονται πολύ μεγάλα.

- **SPIMI**

Βασική διαφορά:

- χρησιμοποιεί term αντί για termid
- κάθε όρος *εισάγεται απευθείας στο posting list* (αντί να έχουμε τα ζεύγη (termid, docid) και διάταξη)

SPIMI: Single-pass in-memory indexing (ευρετηρίαση ενός περάσματος)

Δε διατηρούμε term-termID απεικονίσεις μεταξύ blocks.

Εναλλακτικός αλγόριθμος: Αποφυγή της διάταξης των όρων.

- Συγκεντρώσετε τις καταχωρήσεις σε λίστες καταχωρήσεων όπως αυτές εμφανίζονται.
- Κατασκευή ενός *πλήρους αντεστραμμένου ευρετηρίου και λεξικού* για κάθε block. Χρησιμοποίησε κατακερματισμό (hash) ώστε οι καταχωρήσεις του ίδιου όρου στον ίδιο κάδο
- Μετά συγχωνεύουμε τα ξεχωριστά ευρετήρια σε ένα μεγάλο.

SPIMI-Invert

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

```

Χρησιμοποιούμε *hash* ώστε οι καταχωρήσεις για τον ίδιο όρο στον ίδιο «κάδο»

Το sorting αναφέρεται στο λεξικό

- Η συγχώνευση όπως και στο BSBI.

Δυναμικά ευρετήρια

- Μέχρι στιγμής, θεωρήσαμε ότι τα ευρετήρια είναι *στατικά*.
- Αυτό συμβαίνει σπάνια, στην πραγματικότητα:
 - Νέα έγγραφα εμφανίζονται και πρέπει να ευρετηριοποιηθούν
 - Έγγραφα τροποποιούνται ή διαγράφονται
- Αυτό σημαίνει ότι *πρέπει να ενημερώσουμε τις λίστες καταχωρήσεων*:
 - Αλλαγές στις καταχωρήσεις όρων που είναι ήδη στο λεξικό
 - Προσθήκη νέων όρων στο λεξικό

Μια απλή προσέγγιση

- Διατήρησε ένα «μεγάλο» κεντρικό ευρετήριο
- Τα νέα έγγραφα σε μικρό «βοηθητικό» ευρετήριο (**auxiliary index**) (στη μνήμη)
- Ψάξε και στα δύο, συγχώνευσε το αποτέλεσμα
- Διαγραφές
 - Invalidation bit-vector για τα διαγραμμένα έγγραφα
 - Φιλτράρισμα αποτελεσμάτων ώστε όχι διαγραμμένα
- Περιοδικά, re-index το βοηθητικό στο κυρίως ευρετήριο

Πολυπλοκότητα

- Αποθηκεύουμε κάθε λίστα καταχωρήσεων σε διαφορετικό αρχείο ή όλο το ευρετήριο σε ένα αρχείο;
Έστω σε ένα αρχείο

Έστω T ο συνολικός αριθμός των καταχωρήσεων και n οι καταχωρήσεις που χωρούν στη μνήμη

Κατασκευή

- *Κυρίως και βοηθητικό ευρετήριο*: T/n συγχωνεύσεις, σε κάθε μία κοιτάμε όλους τους όρους, άρα πολυπλοκότητα $O(T^2)$

Ερώτημα

- *Κυρίως και βοηθητικό ευρετήριο*: $O(1)$

Θέματα

- Συχνές συγχωνεύσεις
- Κακή απόδοση κατά τη διάρκεια της συγχώνευσης
- Πιο αποδοτικό αν κάθε λίστα καταχωρήσεων ήταν αποθηκευμένη σε διαφορετικό αρχείο (τότε, απλώς append), αλλά θα χρειαζόμαστε πολλά αρχεία (μη αποδοτικό για το ΛΣ)
- Θα υποθέσουμε ότι όλο το ευρετήριο σε ένα αρχείο.
- Στην πραγματικότητα: Κάτι ανάμεσα (π.χ., πολλές μικρές λίστες καταχώρησης σε ένα αρχείο, διάσπαση πολύ μεγάλων λιστών, κλπ)

Λογαριθμική συγχώνευση

- Διατήρηση μια σειράς από ευρετήρια, το καθένα διπλάσιου μεγέθους από τα προηγούμενα
 - Κάθε στιγμή, χρησιμοποιούνται κάποια από αυτά
- Έστω n ο αριθμός των postings στη μνήμη
- Διατηρούμε στο δίσκο ευρετήρια I_0, I_1, \dots
 - I_0 μεγέθους $2^0 * n$, I_1 μεγέθους $2^1 * n$, I_2 μεγέθους $2^2 * n \dots$
- Ένα βοηθητικό ευρετήριο μεγέθους n στη μνήμη, Z_0

Λογαριθμική συγχώνευση

- Όταν φτάσει το όριο n , τα $2^0 * n$ postings του Z_0 μεταφέρονται στο δίσκο
- Ως ένα νέο index I_0
- Την επόμενη φορά που το Z_0 γεμίζει, συγχώνευση με I_0
- Αποθηκεύεται ως I_1 (αν δεν υπάρχει ήδη I_1) ή συγχώνευση με I_1 ως Z_2 κλπ
- Τα ερωτήματα απαντώνται με χρήση του Z_0 στη μνήμη και όσων I_i υπάρχουν στο δίσκο κάθε φορά

Παράδειγμα, $n = 2$, $T = 15$

LMERGEADDTOKEN(*indexes*, Z_0 , *token*)

```

1   $Z_0 \leftarrow \text{MERGE}(Z_0, \{\text{token}\})$ 
2  if  $|Z_0| = n$ 
3    then for  $i \leftarrow 0$  to  $\infty$ 
4      do if  $l_i \in \text{indexes}$ 
5        then  $Z_{i+1} \leftarrow \text{MERGE}(l_i, Z_i)$ 
6          ( $Z_{i+1}$  is a temporary index on disk.)
7           $\text{indexes} \leftarrow \text{indexes} - \{l_i\}$ 
8        else  $l_i \leftarrow Z_i$  ( $Z_i$  becomes the permanent index  $l_i$ .)
9           $\text{indexes} \leftarrow \text{indexes} \cup \{l_i\}$ 
10         BREAK
11          $Z_0 \leftarrow \emptyset$ 

```

LOGARITHMICMERGE()

```

1   $Z_0 \leftarrow \emptyset$  ( $Z_0$  is the in-memory index.)
2   $\text{indexes} \leftarrow \emptyset$ 
3  while true
4  do LMERGEADDTOKEN(indexes,  $Z_0$ , GETNEXTTOKEN())

```

Πολυπλοκότητες

Κατασκευή

- *Κυρίως και βοηθητικό ευρετήριο*: T/n συγχωνεύσεις, σε κάθε μία κοιτάμε όλους τους όρους, άρα πολυπλοκότητα $O(T^2)$

Το πολύ **$\log T$** indexes, μέγεθος του μεγαλύτερου T

- *Λογαριθμική συγχώνευση*: κάθε καταχώρηση συγχωνεύεται $O(\log T)$ φορές, άρα πολυπλοκότητα $O(T \log T)$

Ερώτημα

- *Κυρίως και βοηθητικό ευρετήριο*: $O(1)$
- *Λογαριθμική συγχώνευση*: κοιτάμε $O(\log T)$ ευρετήρια

Γενικά, περιπλέκεται η ανάκτηση, οπότε συχνά πλήρης ανακατασκευή του ευρετηρίου

Δυναμικά ευρετήρια στις μηχανές αναζήτησης

- Πολύ συχνές αλλαγές
- Συχνά περιοδική *ανακατασκευή του ευρετηρίου από την αρχή*
 - Ενώ κατασκευάζεται το νέο, χρησιμοποιείται το παλιό και όταν η κατασκευή τελειώσει χρήση του νέου

Άλλα θέματα

- Η *διάταξη των εγγράφων στις λίστες* δε γίνεται πάντα με βάση το DocID αλλά μπορεί και με βάση τη συχνότητα εμφάνισης του όρου στο έγγραφο (πιο περίπλοκο γιατί δεν αρκεί append)
- Λίστες δικαιωμάτων προσπέλασης (Access Control Lists ACLs)
 - Για κάθε χρήστη, μια λίστα καταχωρήσεων με τα έγγραφα που μπορεί να προσπελάσει

Κατανεμημένη κατασκευή

- Για ευρετήριο κλίμακας web
Χρήση κατανεμημένου cluster
- Επειδή μια μηχανή είναι επιρρεπής σε αποτυχία
 - Μπορεί απροσδόκητα να γίνει αργή ή να αποτύχει
- Χρησιμοποίηση πολλών μηχανών

MapReduce

ΣΤΑΤΙΣΤΙΚΑ

Στατιστικά

- Πόσο μεγάλο είναι το λεξικό και οι καταχωρήσεις;

BRUTUS	→	1	2	4	11	31	45	173	174	
CAESAR	→	1	2	4	5	6	16	57	132	...
CALPURNIA	→	2	31	54	101					

Λεξιλόγιο και μέγεθος συλλογής

- Πόσο μεγάλο είναι το λεξιλόγιο όρων;
 - Δηλαδή, πόσοι είναι οι διαφορετικοί όροι;
- Υπάρχει *κάποιο άνω όριο* (ή μεγαλώνει συνεχώς με τη προσθήκη νέων εγγράφων);

Π.χ., το Oxford English Dictionary 600,000 λέξεις, αλλά στις πραγματικά μεγάλες συλλογές ονόματα προσώπων, προϊόντων, κλπ

- ✓ Στην πραγματικότητα, το λεξιλόγιο συνεχίζει να μεγαλώνει με το μέγεθος της συλλογής

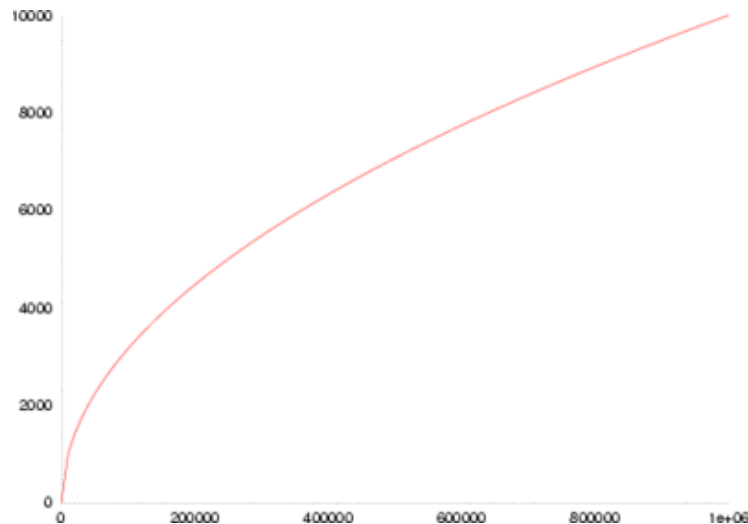
Λεξιλόγιο και μέγεθος συλλογής

Ο νόμος του Heaps:

$$M = k T^b$$

M είναι το μέγεθος του λεξιλογίου (αριθμός όρων), T ο αριθμός των tokens στη συλλογή

- περιγράφει πόσο μεγαλώνει το λεξιλόγιο όσο μεγαλώνει η συλλογή (το συνολικό μήκος των εγγράφων)
- Συνήθης τιμές: $30 \leq k \leq 100$ (εξαρτάται από το είδος της συλλογής) και $b \approx 0.5$



Λεξιλόγιο και μέγεθος συλλογής

- Diminishing returns: μπορούμε γρήγορα να καλύψουμε μέρος του λεξιλογίου, αλλά γίνεται όλο και πιο δύσκολο να το καλύψουμε όλο
- Σε log-log plot του μεγέθους M του λεξιλογίου με το T , ο νόμος προβλέπει γραμμή με κλίση περίπου $\frac{1}{2}$

Για το RCV1, η
διακεκομμένη γραμμή

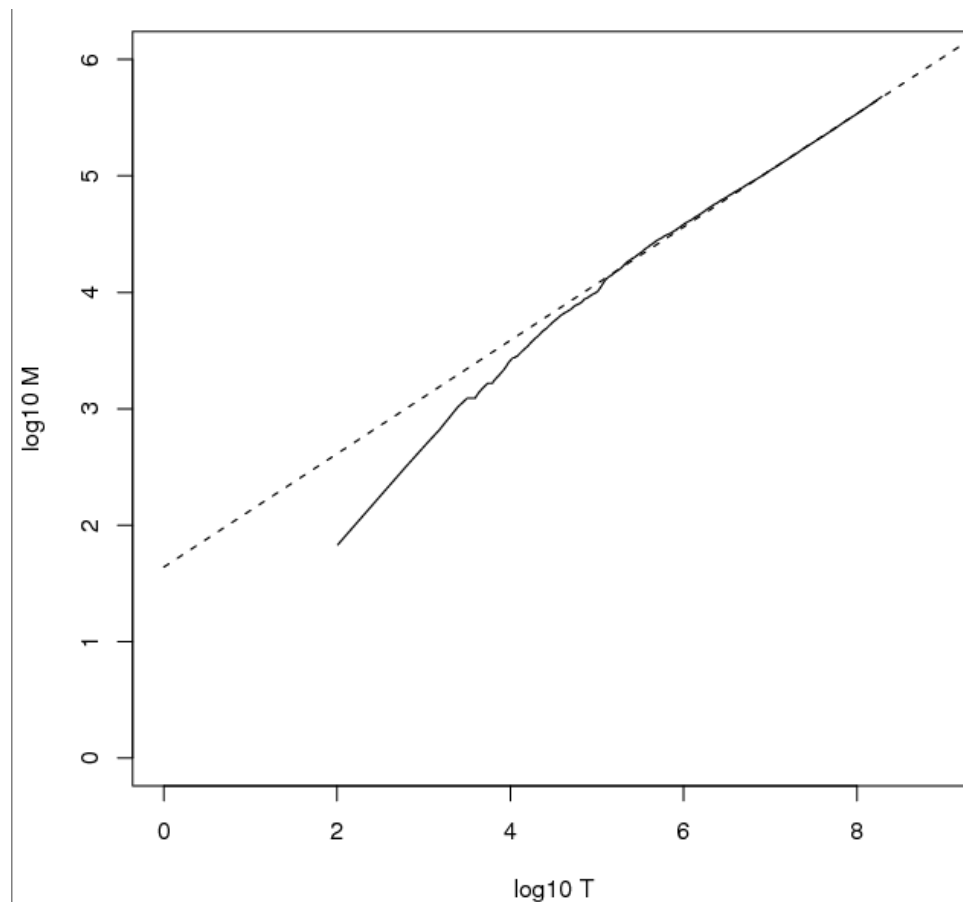
$$\log_{10} M = 0.49 \log_{10} T + 1.64$$

(best least squares fit)

Οπότε, $M = 10^{1.64} T^{0.49}$, άρα
 $k = 10^{1.64} \approx 44$ and $b = 0.49$.

Καλή προσέγγιση για το
Reuters RCV1!

Για το πρώτα **1,000,020**
tokens, ο νόμος προβλέπει
38,323 όρους, στην
πραγματικότητα 38,365



Ο νόμος του Heaps

Τα παρακάτω επηρεάζουν το μέγεθος του λεξικού (και την παράμετρο k):

- Stemming
- Including numbers
- Spelling errors
- Case folding

Ο νόμος του Zipf

✓ Ο νόμος του Heaps μας δίνει το μέγεθος του λεξιλογίου μιας συλλογής (σε συνάρτηση του μεγέθους της συλλογής)

Θα εξετάσουμε τη *σχετική συχνότητα* των όρων

- Στις φυσικές γλώσσες, υπάρχουν **λίγοι** πολύ **συχνοί** όροι και πάρα **πολύ σπάνιοι**

Ο νόμος του Zipf

Ο νόμος του Zipf: Ο i -οστός πιο συχνός όρος έχει συχνότητα ανάλογη του $1/i$.

$$cf_i \propto 1/i = K/i$$

cf_i collection frequency: ο αριθμός εμφανίσεων του όρου t_i στη συλλογή

K μια normalizing constant

Η συχνότητα εμφάνισης ενός όρου είναι αντιστρόφως ανάλογη της θέσης του στη διάταξη με βάση τις συχνότητες

- Αν ο πιο συχνός όρος (ο όρος *the*) εμφανίζεται cf_1 φορές
- Τότε ο δεύτερος πιο συχνός (*of*) εμφανίζεται $cf_1/2$ φορές
- Ο τρίτος (*and*) $cf_1/3$ φορές
- ...

Ο νόμος του Zipf

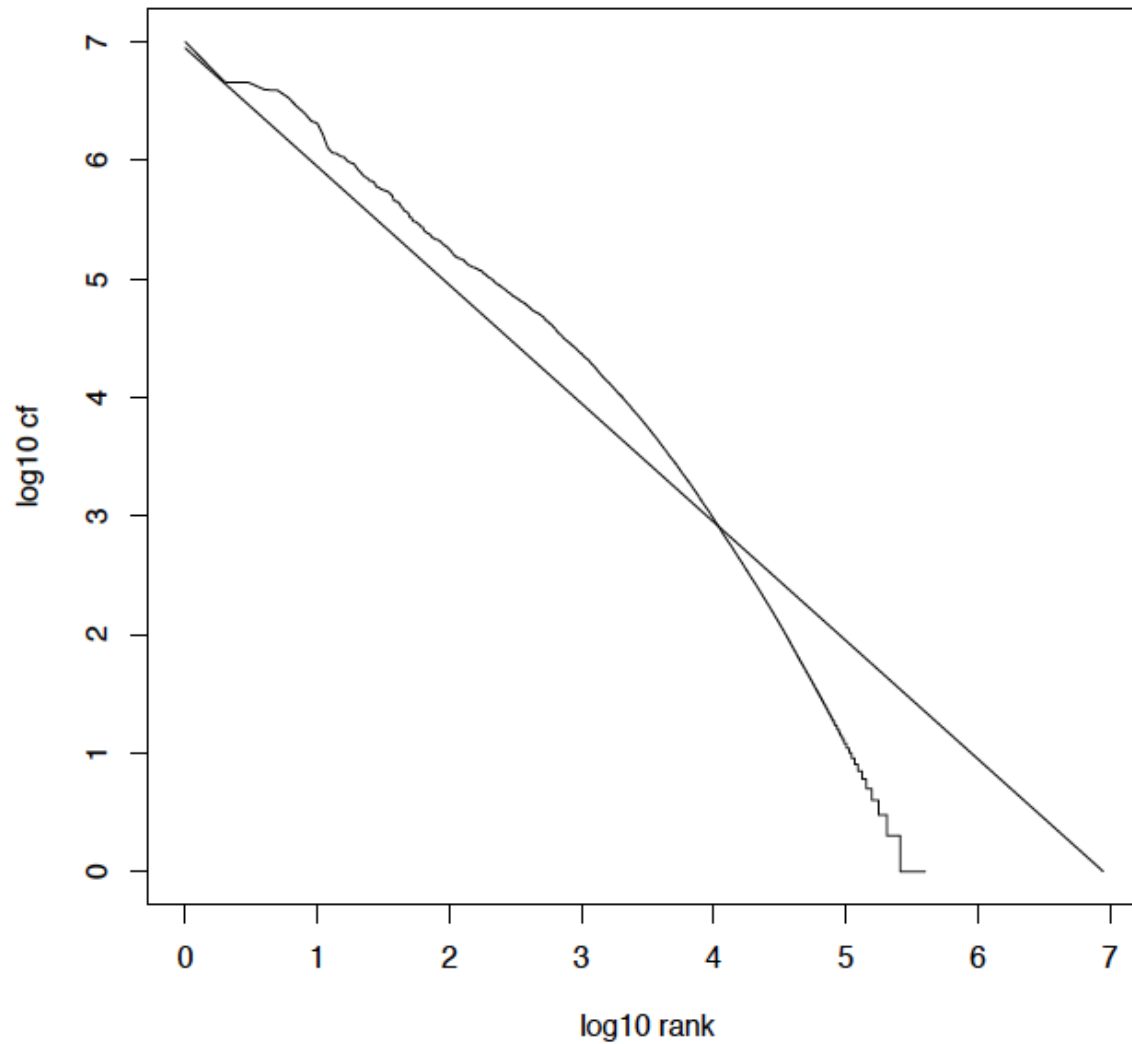
$$cf_i = mi^{-k}$$

$$\log cf_i = \log m - k \log i$$

- Γραμμική σχέση μεταξύ $\log cf_i$ και $\log i$

power law σχέση (εκθετικός νόμος)

Zipf's law for Reuters RCV1



Στατιστικά για τη συλλογή Reuters RCV1

<i>N</i>	documents	800,000
<i>L</i>	tokens per document	200
<i>M</i>	terms (= word types)	400,000
	bytes per token (incl. spaces/punct.)	6
	bytes per token (without spaces/punct.)	4.5
	bytes per term (= word type)	7.5
<i>T</i>	non-positional postings	100,000,000

Μέγεθος ευρετηρίου

size of	word types (terms)			non-positional postings			positional postings		
	dictionary			non-positional index			positional index		
	Size (K)	Δ%	cumul %	Size (K)	Δ %	cumul %	Size (K)	Δ %	cumul %
Unfiltered	484			109,971			197,879		
No numbers	474	-2	-2	100,680	-8	-8	179,158	-9	-9
Case folding	392	-17	-19	96,969	-3	-12	179,158	0	-9
30 stopwords	391	-0	-19	83,390	-14	-24	121,858	-31	-38
150 stopwords	391	-0	-19	67,002	-30	-39	94,517	-47	-52
stemming	322	-17	-33	63,812	-4	-42	94,517	0	-52

ΣΥΜΠΙΕΣΗ

Συμπίεση

- Θα δούμε μερικά θέματα για τη συμπίεση το λεξικού και των λιστών καταχωρήσεων
- Βασικό Boolean ευρετήριο, χωρίς πληροφορία θέσης κλπ

Γιατί συμπίεση;

- Λιγότερος χώρος στη μνήμη
 - Λίγο πιο οικονομικό
- Κρατάμε περισσότερα πράγματα στη μνήμη
 - Αύξηση της ταχύτητας
- Αύξηση της ταχύτητας μεταφοράς δεδομένων από το δίσκο στη μνήμη
 - [διάβασε τα συμπιεσμένα δεδομένα | αποσυμπίεσε] γρηγορότερο από [διάβασε μη συμπιεσμένα δεδομένα]
 - Προϋπόθεση: Γρήγοροι αλγόριθμοι αποσυμπίεσης

Απωλεστική και μη συμπίεση

- **Lossless compression**: (μη απωλεστική συμπίεση) Διατηρείτε όλη η πληροφορία
 - Αυτή που κυρίως χρησιμοποιείται σε ΑΠ
- **Lossy compression**: (απωλεστική συμπίεση) Κάποια πληροφορία χάνεται
 - Πολλά από τα **βήματα προ-επεξεργασίας** (μετατροπή σε μικρά, stop words, stemming, number elimination) μπορεί να θεωρηθούν ως απωλεστική συμπίεση
 - Μπορεί να είναι αποδεκτή στην περίπτωση π.χ., που μας ενδιαφέρουν μόνο τα κορυφαία από τα σχετικά έγγραφα

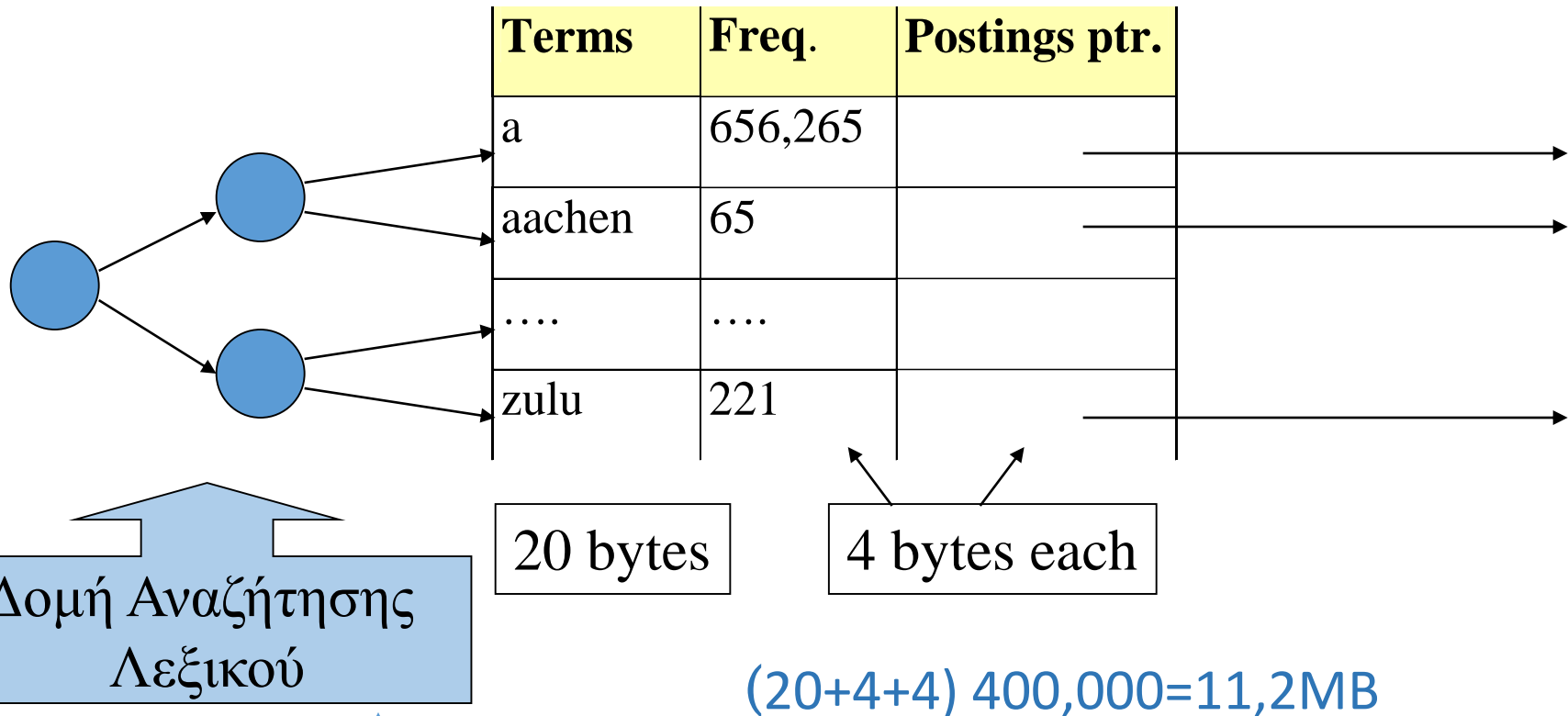
ΣΥΜΠΙΕΣΗ ΛΕΞΙΚΟΥ

Συμπύεση λεξικού

- Η αναζήτηση αρχίζει από το λεξικό -> Θα θέλαμε να το κρατάμε στη μνήμη
- Συνυπάρχει με άλλες εφαρμογές (memory footprint competition)
- Κινητές/ενσωματωμένες συσκευές μικρή μνήμη
- Ακόμα και αν όχι στη μνήμη, θα θέλαμε να είναι μικρό για γρήγορη αρχή της αναζήτησης

Αποθήκευση λεξικού

- Κάθε εγγραφή: τον όρο, συχνότητα εμφάνισης, δείκτη
- Θα θεωρήσουμε **την πιο απλή αποθήκευση**, ως ταξινομημένο πίνακα εγγραφών σταθερού μεγέθους (array of fixed-width entries)
 - ~400,000 όροι; 28 bytes/term = 11.2 MB.



Δομή Αναζήτησης
Λεξικού

Θα την αγνοήσουμε ↗

Αποθήκευση λεξικού

Σπατάλη χώρου

- Πολλά από τα bytes στη στήλη **Term** δεν χρησιμοποιούνται – δίνουμε 20 bytes για όρους με 1 χαρακτήρα
 - Και δε μπορούμε να χειριστούμε το *supercalifragilisticexpialidocious* ή *hydrochlorofluorocarbons* (λέξεις με πάνω από 20 χαρακτήρες)
- Μέσος όρος στο γραπτό λόγο για τα Αγγλικά είναι ~4.5 χαρακτήρες/λέξη.
- Μέσος όρος των λέξεων στο λεξικό για τα Αγγλικά: ~8 χαρακτήρες
- Οι μικρές λέξεις κυριαρχούν στα tokens αλλά όχι στους όρους.

Συμπύεση της λίστας όρων: Λεξικό-ως-Σειρά-Χαρακτήρων

Αποθήκευσε το λεξικό ως ένα (μεγάλο) string χαρακτήρων:

- ❖ Ένας δείκτης δείχνει στο τέλος της τρέχουσας λέξης (αρχή επόμενης)
- ❖ Εξοικονόμηση **60%** του χώρου

....systilesyzygeticsyzygialsyzygyszaibelyiteszczecinszomo....

Freq.	Postings ptr.	Term ptr.
33		
29		
44		
126		

Συνολικό μήκος της σειράς (string) =
400K x 8B = 3.2MB

Δείκτες για 3.2M
θέσεις: $\log_2 3.2M =$
22bits = 3bytes

δυναμική αναζήτηση όπως πριν, τώρα στο string

Χώρος για το λεξικό ως string

- 4 bytes ανά όρο για το **Freq.**
 - 4 bytes ανά όρο για δείκτες σε **Postings.**
 - 3 bytes ανά **term pointer**
- } Κατά μέσο όρο: **11** bytes /term
- Κατά μέσο όρο **8** bytes ανά όρο στο string (3.2MB)
 - 400K όροι x **19** \Rightarrow 7.6 MB (έναντι 11.2MB για σταθερό μήκος λέξης)

Blocking (Δείκτες σε ομάδες)

- Διαίρεσε το string σε ομάδες (blocks) των k όρων
- Διατήρησε ένα δείκτη σε κάθε ομάδα
 - Παράδειγμα: $k = 4$.
- Χρειαζόμαστε και το **μήκος** του όρου (1 extra byte)

....7systile9syzygetic8syzygial6syzygy11szaiibelyite8szczecin9szomo....

Freq.	Postings ptr.	Term ptr.
33		
29		
44		
126		
7		

Κερδίζουμε 3 bytes
για $k - 1$
δείκτες.

Χάνουμε 4 (k) bytes για
το μήκος του όρου

Blocking

Συνολικό όφελος για block size $k = 4$

- Χωρίς blocking 3 bytes/pointer
 - $3 \times 4 = 12$ bytes, (ανά block)

Τώρα $3 + 4 = 7$ bytes.

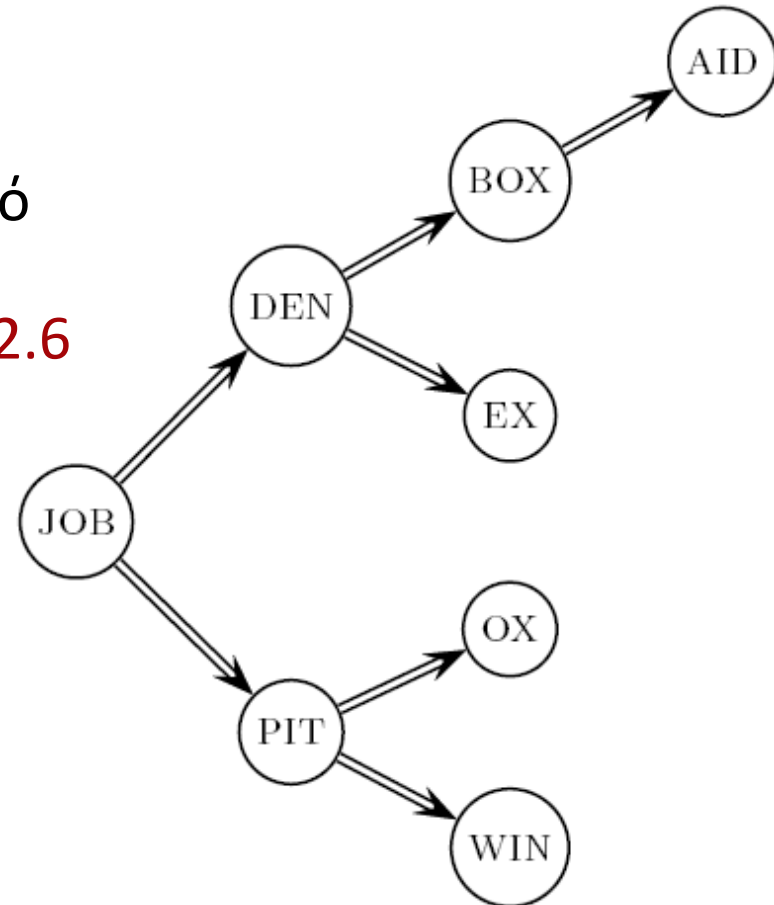
Εξοικονόμηση ακόμα ~ 0.5 MB. Ελάττωση του μεγέθους του ευρετηρίου από 7.6 MB σε 7.1 MB.

- Γιατί όχι ακόμα μεγαλύτερο k ;
- Σε τι χάνουμε;

Αναζήτηση στο λεξικό χωρίς Blocking

- Ας υποθέσουμε *δυναμική αναζήτηση* και ότι κάθε όρος ισοπίθανο να εμφανιστεί στην ερώτηση (όχι και τόσο ρεαλιστικό στη πράξη) μέσος αριθμός συγκρίσεων = $(1+2\cdot 2+4\cdot 3+4)/8 \sim 2.6$

Άσκηση: σκεφτείτε ένα καλύτερο τρόπο αναζήτησης αν δεν έχουμε ομοιόμορφη κατανομή των όρων

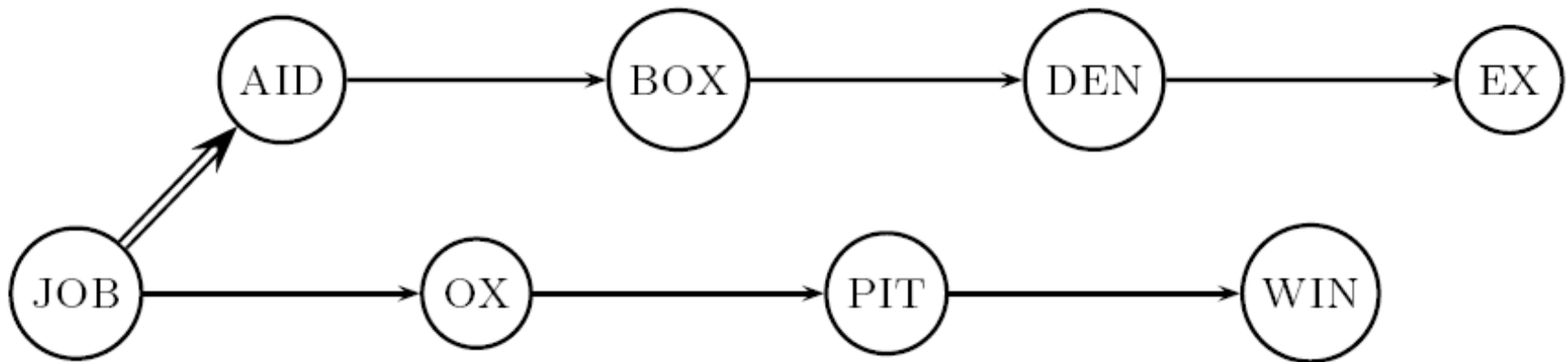


Αναζήτηση στο λεξικό με Blocking

Διαδική αναζήτηση μας οδηγεί σε ομάδες (block) από $k = 4$ όρους

Μετά γραμμική αναζήτηση στους $k = 4$ αυτούς όρους.

Μέσος όρος $(1+2\cdot 2+2\cdot 3+2\cdot 4+5)/8 = 3$



Εμπρόσθια κωδικοποίηση (Front coding)

Οι λέξεις συχνά έχουν μεγάλα κοινά προθέματα – αποθήκευση μόνο των διαφορών

8automata8automate9automatic10automation

→ *8automat* a1 e2 ic3 ion*

Encodes *automat*

Extra length
beyond *automat*.

Εμπρόσθια κωδικοποίηση (Front coding)

Αν στο δίσκο, μπορούμε να έχουμε ένα B-δέντρο με τον πρώτο όρο σε κάθε σελίδα

Κατακερματισμός ελαττώνει το μέγεθος αλλά πρόβλημα με ενημερώσεις

Η συλλογή RCV1: στατιστικά

N	documents	800,000
L	tokens per document	200
M	terms (= word types)	400,000
	bytes per token (incl. spaces/punct.)	6
	bytes per token (without spaces/punct.)	4.5
	bytes per term (= word type)	7.5
T	non-positional postings	100,000,000

- *Γιατί κατά μέσο ένα term είναι μεγαλύτερο από ένα token;*

Περίληψη συμπίεσης για το λεξικό του RCV1

Τεχνική	Μέγεθος σε MB
Fixed width	11.2
Dictionary-as-String with pointers to every term	7.6
Also, blocking $k = 4$	7.1
Also, Blocking + front coding	5.9

ΣΥΜΠΙΕΣΗ ΤΩΝ ΚΑΤΑΧΩΡΗΣΕΩΝ

Συμπίεση των καταχωρήσεων

- Το αρχείο των καταχωρήσεων είναι *πολύ μεγαλύτερο* αυτού του λεξικού - τουλάχιστον 10 φορές.
- Βασική επιδίωξη: *αποθήκευση κάθε καταχώρησης συνοπτικά*
- Στην περίπτωση μας, μια καταχώρηση είναι το αναγνωριστικό ενός εγγράφου (**docID**).
 - Για τη συλλογή του Reuters (800,000 έγγραφα), μπορούμε να χρησιμοποιήσουμε 32 bits ανά docID αν έχουμε ακεραίους 4-bytes.
 - Εναλλακτικά, $\log_2 800,000 \approx 20$ bits ανά docID.
- Μπορούμε λιγότερο από 20 bits ανά docID;

Συμπύεση των καταχωρήσεων

- Μέγεθος της συλλογής
 - $800,000 \text{ (έγγραφα)} \times 200 \text{ (token)} \times 6 \text{ bytes} = 960 \text{ MB}$
- Μέγεθος του αρχείου καταχωρήσεων
 - $100,000,000 \text{ (καταχωρήσεις)} \times 20/8 \text{ bytes} = 250 \text{ MB}$

Συμπύεση των καταχωρήσεων

- Αποθηκεύουμε τη λίστα των εγγράφων σε αύξουσα διάταξη των docID.
 - **computer**: 33, 47, 154, 159, 202 ...
- Συνέπεια: αρκεί να αποθηκεύουμε τα διάκενα (*gaps*).
 - 33, 14, 107, 5, 43 ...
- Γιατί: Τα περισσότερα διάκενα μπορεί να κωδικοποιηθούν/αποθηκευτούν με πολύ λιγότερα από 20 bits.

Παράδειγμα

	encoding	postings list					
THE	docIDs	...	283042	283043	283044	283045	...
	gaps		1	1	1		...
COMPUTER	docIDs	...	283047	283154	283159	283202	...
	gaps		107	5	43		...
ARACHNOCENTRIC	docIDs	252000	500100				
	gaps	252000	248100				

Παρόμοια ιδέα και για positional indexes (κωδικοποίηση των κενών ανάμεσα στις θέσεις)

Συμπύεση των καταχωρήσεων

- Ένας όρος όπως **arachnocentric** εμφανίζεται ίσως σε ένα έγγραφο στο εκατομμύριο.
- Ένας όρος όπως **the** εμφανίζεται σχεδόν σε κάθε έγγραφο, άρα 20 bits/εγγραφή πολύ ακριβό

Κωδικοποίηση μεταβλητού μεγέθους (Variable length encoding)

Στόχος:

- Για το *arachnocentric*, θα χρησιμοποιήσουμε εγγραφές ~ 20 bits/gap.
- Για το *the*, θα χρησιμοποιήσουμε εγγραφές ~ 1 bit/gap entry.
- Αν το μέσο κενό για έναν όρο είναι G , θέλουμε να χρησιμοποιήσουμε εγγραφές $\sim \log_2 G$ bits/gap.
- Βασική πρόκληση: κωδικοποίηση κάθε ακεραίου (gap) *με όσα λιγότερα bits* είναι απαραίτητα για αυτόν τον ακέραιο.
- Αυτό απαιτεί κωδικοποίηση μεταβλητού μεγέθους -- *variable length encoding*
- Αυτό το πετυχαίνουμε χρησιμοποιώντας σύντομους κώδικες για μικρούς αριθμούς

Κωδικοί μεταβλητών Byte (Variable Byte (VB) codes)

- Κωδικοποιούμε κάθε διάκενο με ακέραιο αριθμό από bytes
- Το πρώτο bit κάθε byte χρησιμοποιείται ως *bit συνέχισης* (continuation bit)
 - 0, αν ακολουθεί και άλλο byte
 - 1, αλλιώς (αν το τελευταίο)
 - Είναι 0 σε όλα τα bytes εκτός από το τελευταίο, όπου είναι 1
- Χρησιμοποιείται για να σηματοδοτήσει το τελευταίο byte της κωδικοποίησης

Κωδικοί μεταβλητών Byte (Variable Byte (VB) codes)

- Ξεκίνα με ένα byte για την αποθήκευση του G
- Αν $G \leq 127$, υπολόγισε τη δυαδική αναπαράσταση με τα 7 διαθέσιμα bits and θέσε $c = 1$
- Αλλιώς, κωδικοποίησε τα 7 lower-order bits του G και χρησιμοποίησε επιπρόσθετα bytes για να κωδικοποιήσεις τα higher order bits με τον ίδιο αλγόριθμο
- Στο τέλος, θέσε το bit συνέχισης του τελευταίου byte σε 1, $c = 1$ και στα άλλα σε 0, $c = 0$.

Παράδειγμα

docIDs	824	829	215406
gaps		5	214577
VB code	00000110 10111000	10000101	00001101 00001100 10110001

Postings stored as the byte concatenation

000001101011100010000101000011010000110010110001

Key property: VB-encoded postings are uniquely *prefix-decodable*.

824
1100111000

For a small gap (5), VB uses a whole byte.

Άλλες κωδικοποιήσεις

- Αντί για bytes, δηλαδή 8 bits, άλλες μονάδες πχ 32 bits (words), 16 bits, 4 bits (nibbles).

Compression ratio vs speed of decompression

- Με byte χάνουμε κάποιο χώρο αν πολύ μικρά διάκενα – nibbles καλύτερα σε αυτές τις περιπτώσεις.
 - Μικρές λέξεις, πιο περίπλοκος χειρισμός
- Οι κωδικοί VB χρησιμοποιούνται σε πολλά εμπορικά/ερευνητικά συστήματα

Συμπίεση του RCV1

Data structure	Size in MB
dictionary, fixed-width	11.2
dictionary, term pointers into string	7.6
with blocking, $k = 4$	7.1
with blocking & front coding	5.9
collection (text, xml markup etc)	3,600.0
collection (text)	960.0
Term-doc incidence matrix	40,000.0
postings, uncompressed (32-bit words)	400.0
postings, uncompressed (20 bits)	250.0
postings, variable byte encoded	116.0
<i>postings, γ-encoded</i>	<i>101.0</i>

Συμπεράσματα

- Μπορούμε να κατασκευάσουμε ένα ευρετήριο για Boolean ανάκτηση πολύ αποδοτικό από άποψη χώρου
- Μόνο 4% του συνολικού μεγέθους της συλλογής
- Μόνο το 10-15% του συνολικού κειμένου της συλλογής
- Βέβαια, έχουμε αγνοήσει την **πληροφορία θέσης (positional indexes)**
 - Η εξοικονόμηση χώρου είναι μικρότερη στην πράξη
 - Αλλά, οι τεχνικές είναι παρόμοιες – χρησιμοποίηση gaps και για τις θέσεις στο έγγραφο

ΤΕΛΟΣ 4^{ου}-5^{ου} Κεφαλαίου

Ερωτήσεις?

Χρησιμοποιήθηκε κάποιο υλικό των:

✓ *Pandu Nayak and Prabhakar Raghavan, CS276: Information Retrieval and Web Search (Stanford)*