Introduction to

Information Retrieval

ΠΛΕ70: Ανάκτηση Πληροφορίας

Διδάσκουσα: Ευαγγελία Πιτουρά

Διάλεξη 5: Κατασκευή Ευρετηρίου. Στατιστικά Συλλογής.

Τι είδαμε στο προηγούμενο μάθημα

Ανάκτηση Ανεκτική σε Σφάλματα

- Διόρθωση λαθών

Διόρθωση ορθογραφικών λαθών

- Δύο βασικές χρήσεις
  - Διόρθωση των *εγγράφων* που ευρετηριοποιούνται
  - Διόρθωση των *ερωτημάτων* ώστε να ανακτηθούν «σωστές» απαντήσεις
- Δυο βασικές κατηγορίες:
  - *Μεμονωμένες λέξεις*
    - Εξέτασε κάθε λέξη μόνη της για λάθη
    - Δεν πιάνει typos που έχουν ως αποτέλεσμα σωστά γραμμένες λέξεις
      - π.χ., ***from*** → ***form***
  - Βασισμένη *σε συμφραζόμενα* (context sensitive)
    - Κοιτά στις λέξεις γύρω,
      - π.χ., ***I flew form Heathrow to Narita.***

Διόρθωση μεμονωμένης λέξης

- Θεμελιώδης υπόθεση – *υπάρχει ένα λεξικό που μας δίνει τη σωστή ορθογραφία*
- Δυο βασικές επιλογές για αυτό το λεξικό
  - Ένα standard λεξικό
  - Το λεξικό της συλλογής (corpus)

Δοθέντος ενός **λεξικού** και μιας ακολουθίας χαρακτήρων Q, επέστρεψε τις λέξεις του λεξικού που είναι πιο κοντά στο Q

Εξετάσαμε δύο ορισμούς εγγύτητας:

- Την *απόσταση διόρθωσης (edit distance)* και την *σταθμισμένη* εκδοχή της
- *Επικάλυψη (overlap) n-γραμμάτων*

Απόσταση διόρθωσης (Edit distance)

ΟΡΙΣΜΟΣ: Δοθέντων δυο αλφαριθμητικών (strings) S_1 and S_2, ο ελάχιστος αριθμός πράξεων για τη μετατροπή του ενός στο άλλο

Υπολογισμός με χρήση δυναμικού προγραμματισμού:

Ο βέλτιστος τρόπος από μια λέξη σε μια άλλη, βασίζεται στο βέλτιστο τρόπο από κάποιο πρόθεμα της πρώτης σε πρόθεμα της δεύτερης

Έναν Πίνακα

- ✓ Γραμμές: Γράμματα (προθέματα) της πρώτη λέξης
- ✓ Στήλες: Γράμματα (προθέματα) της δεύτερης λέξης
- ✓ Θέσεις του πίνακα: βέλτιστο κόστος (απόσταση)

Υπολογισμός απόστασης διόρθωσης

String s_2

String s_1

		f	a	s	t
	0	1	2	3	4
c	1	1	2	3	4
a	2	2	1	2	3
t	3	3	2	2	2
s	4	4	3	2	3

cats – fast

Κάθε στοιχείο m[i, j] του πίνακα μας δίνει το βέλτιστο κόστος (απόσταση) για να πάμε από το πρόθεμα μήκους **i** του s_1 στο πρόθεμα μήκους **j** του s_2

Υπολογισμός απόστασης διόρθωσης

Κόστος από τον πάνω αριστερό γείτονα **Copy ή Replace**	Κόστος από τον πάνω γείτονα **Delete**
Κόστος από τον αριστερό γείτονα **Insert**	Το μικρότερο από τα 3 κόστη

Υπολογισμός απόστασης: παράδειγμα

		s	n	o	w
	0	1 1	2 2	3 3	4 4
o	1 1	1 2 2 1	2 3 2 2	2 4 3 2	4 5 3 3
s	2 2	1 2 3 1	2 3 2 2	3 3 3 3	3 4 4 3
l	3 3	3 2 4 2	2 3 3 2	3 4 3 3	4 4 4 4
o	4 4	4 3 5 3	3 3 4 3	2 4 4 2	4 5 3 3

Χρήση των αποστάσεων διόρθωσης

1. Δοθείσας μιας ερώτησης, πρώτα απαρίθμησε όλες τις ακολουθίες χαρακτήρων μέσα σε μια προκαθορισμένη (σταθμισμένη) απόσταση διόρθωσης (π.χ., 2)
2. Βρες την τομή αυτού του συνόλου με τις «σωστές» λέξεις
3. *Πρότεινε τους όρους* που βρήκες στο χρήστη

Εναλλακτικά,

- Ψάξε όλες τις πιθανές διορθώσεις στο αντεστραμμένο ευρετήριο και επέστρεψε όλα τα έγγραφα … αργό
- Μπορούμε να *επιστρέψουμε τα έγγραφα* μόνο για την πιο πιθανή διόρθωση

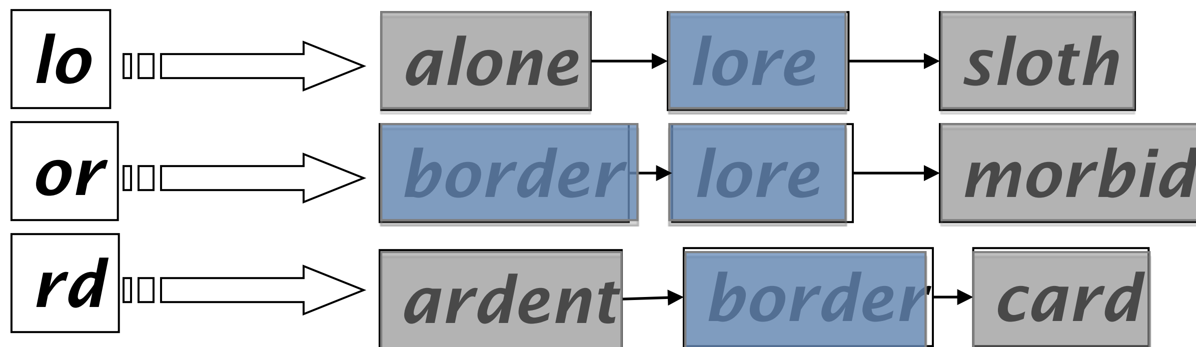
- Η εναλλακτική λύση παίρνει τον έλεγχο από το χρήστη αλλά κερδίζουμε ένα γύρο διάδρασης

Επικάλυψη *k*-γραμμάτων

Εναλλακτικός ορισμός απόστασης: βάση των κοινών *k*-γραμμάτων

1. Απαρίθμησε όλα το *k*-γράμματα στον όρο της ερώτησης
2. Χρησιμοποίησε το ευρετήριο *k*-γραμμάτων για να ανακτήσεις όλους τους όρους του λεξικού που ταιριάζουν κάποιο από τα *k*-γράμματα του ερωτήματος
3. Ανέκτησε όλους τους όρους του λεξικού που ταιριάζουν *κάποιο* (>= *κατώφλι*) αριθμό από τα *k*-γράμματα του ερωτήματος ή Jaccard distance > τιμή

- Παράδειγμα: *k* = 2, ερώτημα lord, απόσταση από lore και border



Διόρθωση εξαρτώμενη από το περιβάλλον

Κείμενο: ***I flew from Heathrow to Narita.***

- Θεωρείστε το ερώτημα-φράση ***"flew form Heathrow"***
- Θα θέλαμε να απαντήσουμε

  Did you mean "***flew from Heathrow***"?

Γιατί δεν υπάρχουν (αρκετά) έγγραφα που να ταιριάζουν στο αρχικό ερώτημα φράση

Διόρθωση βασισμένη στα συμφραζόμενα

- Χρειάζεται συμφραζόμενο περιβάλλον για να το πιάσει αυτό.

Πρώτη ιδέα:

1. Ανέκτησε τους όρους του λεξικού που είναι κοντά (σε σταθμισμένη απόσταση διόρθωσης) από κάθε όρο του ερωτήματος
2. Δοκίμασε όλες τις πιθανές φράσεις που προκύπτουν κρατώντας κάθε φορά μια λέξη σταθερή
   - ***flew from heathrow***
   - ***fled form heathrow***
   - ***flea form heathrow***
3. *Hit-based spelling correction*: Πρότεινε την εναλλακτική με τα περισσότερα hits

Διόρθωση βασισμένη στα συμφραζόμενα

Εναλλακτική Προσέγγιση με χρήση biwords

1. Σπάσε της φράση σε σύζευξη biwords.
2. Ψάξε τα biwords που χρειάζονται διόρθωση μόνο ενός όρου.
3. Απαρίθμησε μόνο τις φράσεις που περιέχουν «κοινά» biwords.

Γενικά θέματα

- Θέλουμε να δούμε διαφορετικές απαντήσεις στο “Did you mean?”
- Ποιες θα επιλέξουμε να παρουσιάσουμε στο χρήστη;
  - Αυτή που εμφανίζεται στα περισσότερα έγγραφα
  - Ανάλυση του Query log

Soundex

Φωνητική διόρθωση: ερώτημα που «ακούγεται» όπως ο σωστός όρος

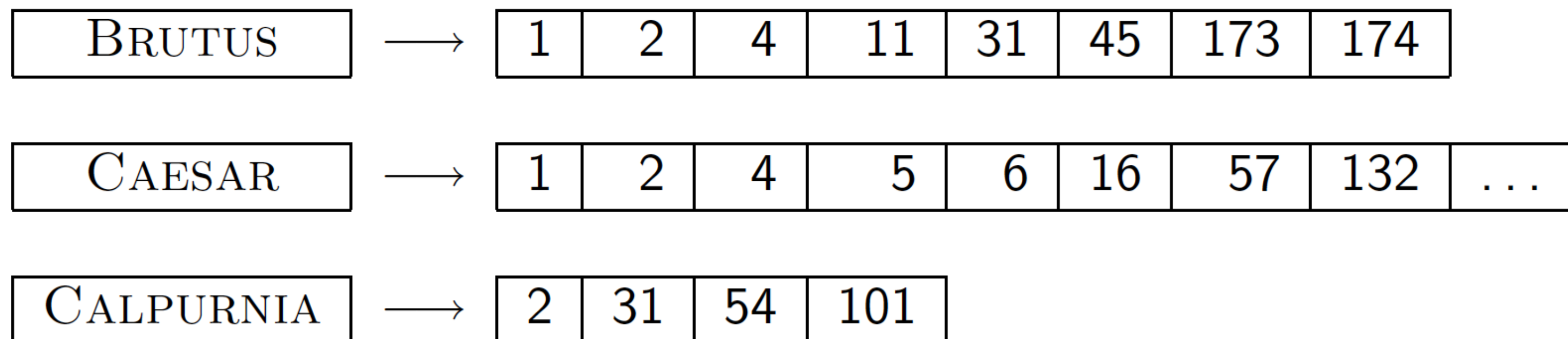
- Κλάση ευριστικών για την επέκταση ενός ερωτήματος σε φωνητικά (phonetic) ισοδύναμα
  - Εξαρτώνται από τη γλώσσα – κυρίως για ονόματα
    - Π.χ., ***chebyshev*** $\rightarrow$ ***tchebycheff***

Τι θα δούμε σήμερα

- Κατασκευή ευρετηρίου
- Στατιστικά για τη συλλογή

ΚΑΤΑΣΚΕΥΗ ΕΥΡΕΤΗΡΙΟΥ

Η βασική δομή: Το αντεστραμμένο ευρετήριο (inverted index)



dictionary

postings

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;
- Ποιες στρατηγικές χρησιμοποιούμε όταν έχουμε περιορισμένη μνήμη;

Βασικά στοιχεία του υλικού

- Πολλές αποφάσεις στην ανάκτηση πληροφορίας βασίζονται στα χαρακτηριστικά του υλικού
- 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)
- 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 – buffer
- Παράλληλα με την επεξεργασία δεδομένων
- Μέγεθος Block: 8KB - 256 KB.

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 μs = 2×10^{-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

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**

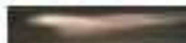
Go to a Section: U.S. International Business Markets Politics Entertainment Technology Sports Oddly Enoug

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.

Ένα έγγραφο της συλλογής Reuters RCV1

Symbol	Statistic	Value
N	documents	800,000
L_{ave}	avg. # tokens per document	200
M	terms	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
T	tokens	100,000,000

Κατασκευή ευρετηρίου

Επεξεργαζόμαστε τα έγγραφα για να βρούμε τις λέξεις - αυτές αποθηκεύονται μαζί 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

Βασικό βήμα: sort

- Αφού έχουμε επεξεργαστεί όλα τα έγγραφα, το αντεστραμμένο ευρετήριο διατάσσεται (sort) με βάση τους όρους

Θα επικεντρωθούμε στο βήμα διάταξης
Πρέπει να διατάξουμε 100M όρους.

Στη συνέχεια, για κάθε όρο, διάταξη εγγράφων

Χρήση termID αντί term

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

Κλιμάκωση της κατασκευής του ευρετηρίου

- 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;
  - Λαμβάνοντας υπ' όψιν τα περιορισμούς και τα χαρακτηριστικά του υλικού. . .

Κατασκευή με βάση τη διάταξη

- Καθώς κατασκευάζουμε το ευρετήριο, επεξεργαζόμαστε τα έγγραφα ένα-ένα
- Οι τελικές καταχωρήσεις για κάθε όρο είναι ημιτελής μέχρι το τέλος

Μπορούμε να κρατάμε όλο το ευρετήριο στη μνήμη;

- Κάθε εγγραφή καταχώρησης (ακόμα και χωρίς πληροφορία θέσης - non-positional) δηλαδή *(term, doc, freq)* καταλαμβάνει 4+4+4 = 12 bytes και απαιτεί πολύ χώρο για μεγάλες συλλογές
- T = 100,000,000 όροι για το RCV1
  - 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

Διάταξη χρησιμοποιώντας το δίσκο σαν «μνήμη»;

- Μπορούμε να χρησιμοποιήσουμε *τον ίδιο αλγόριθμο* κατασκευής για το ευρετήριο αλλά χρησιμοποιώντας δίσκο αντί για μνήμη;

- Όχι: Διάταξη T = 100,000,000 εγγραφών στο δίσκο είναι πολύ αργή – πολλές τυχαίες ανακτήσεις (disk seeks).
- Χρειαζόμαστε έναν αλγόριθμο *εξωτερικής διάταξης (external sorting)*.

Γιατί όχι;

- Διάσχιση του εγγράφου και κατασκευή εγγραφών καταχωρήσεων για ένα έγγραφο τη φορά
- Μετά διάταξη των εγγραφών με βάση τους όρους (και μετά, για κάθε όρο, διάταξη καταχωρήσεων με βάση το έγγραφο)
- Αυτή η διαδικασία με τυχαία ανάκτηση στο δίσκο θα ήταν πολύ αργή – διάταξη *T*=100M εγγραφών

Αν κάθε σύγκριση χρειάζεται 2 προσπελάσεις στο δίσκο, και για τη διάταξη N στοιχείων χρειαζόμαστε $N \log_2 N$ συγκρίσεις, πόσο χρόνο θα χρειαζόμασταν;

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

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

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

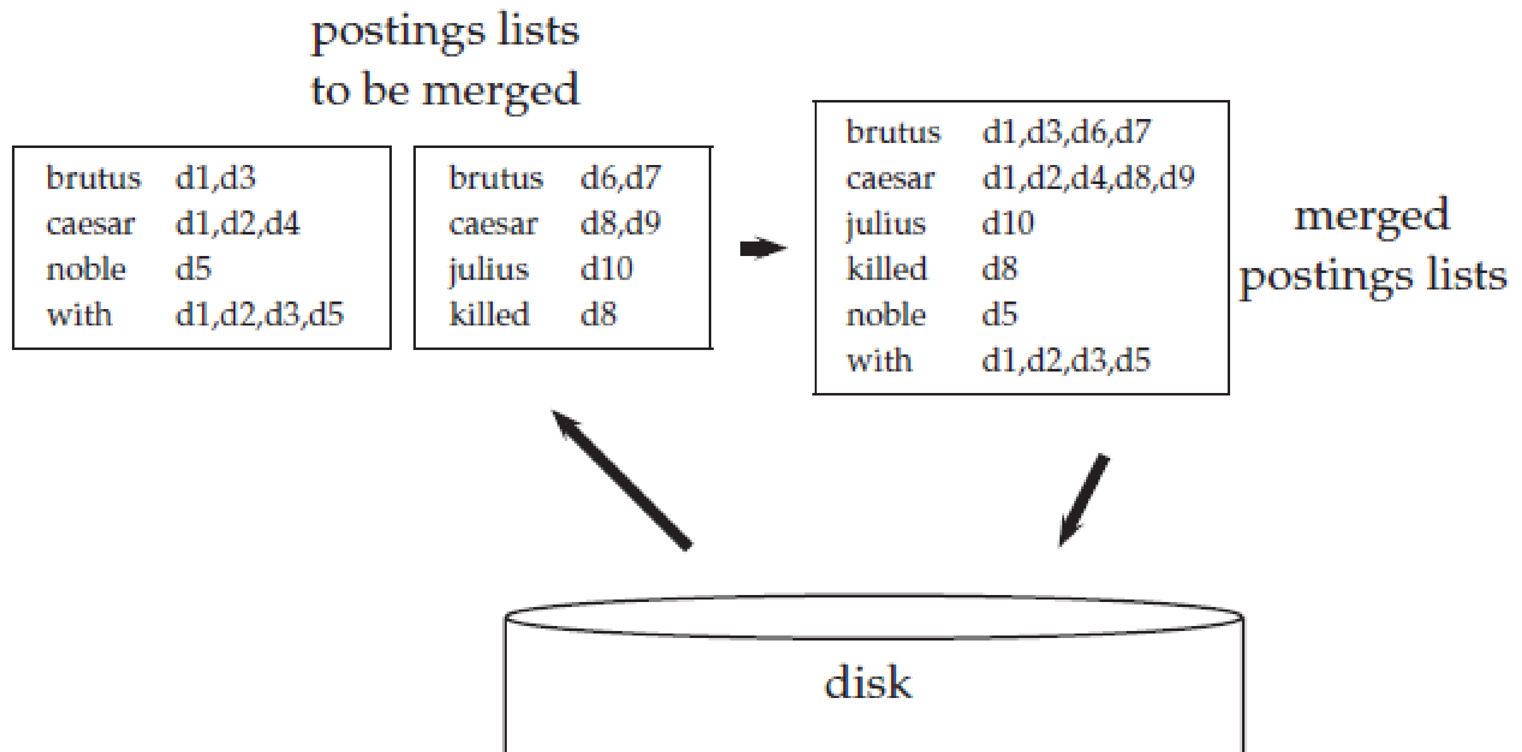
- Εγγραφές 12-byte (4+4+4) *(term, doc, freq).*
- Παράγονται κατά τη διάσχιση των εγγράφων
- Διάταξη 100M τέτοιων 12-byte εγγραφών με βάση τον όρο.
- Ορίζουμε ένα Block ~ 10M τέτοιες εγγραφές
  - Μπορούμε εύκολα να έχουμε κάποια από αυτά στη μνήμη.
  - Αρχικά, 10 τέτοια blocks.
- Βασική ιδέα:
  - Συγκέντρωσε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

Διάταξη 10 blocks των 10M εγγραφών

```
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)
```

Διάβασε ένα-ένα τα έγγραφα – γεμίζοντας ένα block με <term, docid>, διάταξη του block, γράψε το γεμάτο block στο δίσκο

Παράδειγμα

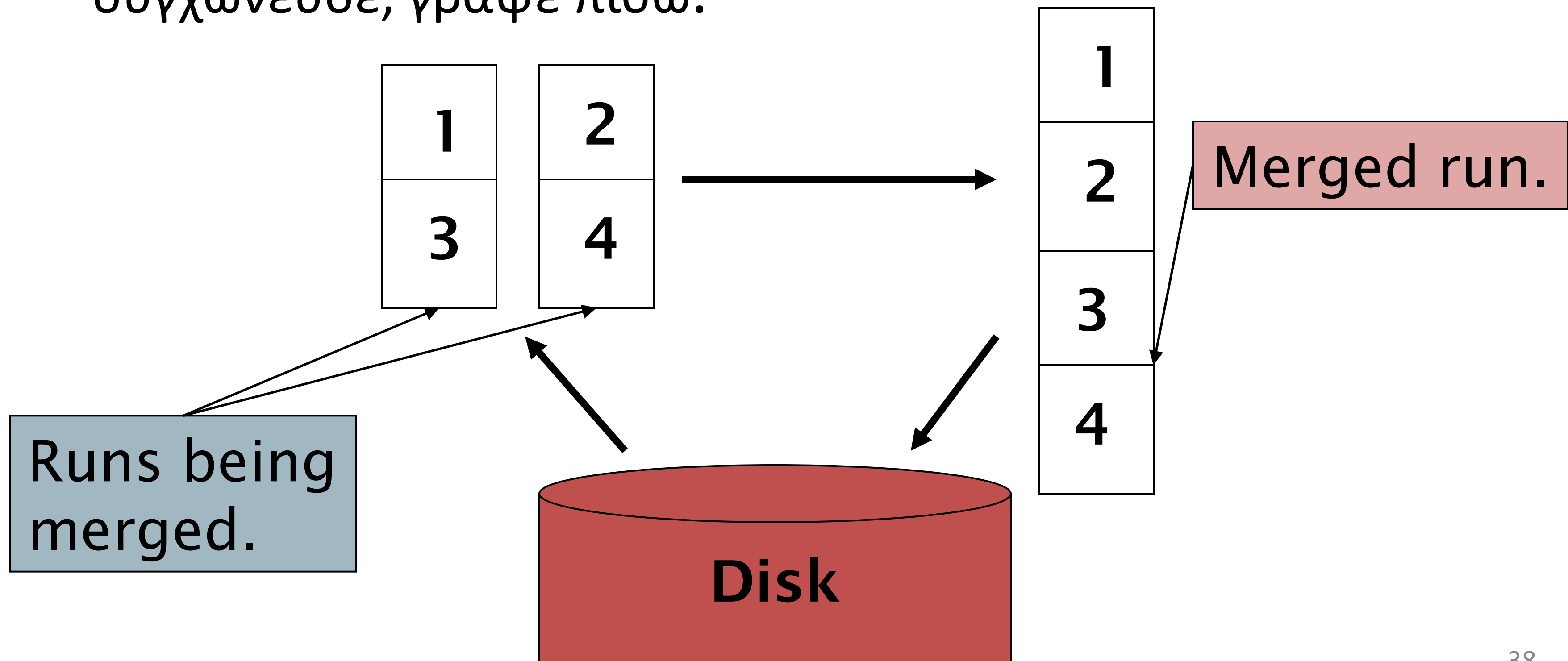


Διάταξη 10 blocks των 10M εγγραφών

- Πρώτα, διάβασε κάθε block και διάταξε τις εγγραφές του:
  - Quicksort 2N ln N αναμενόμενα βήματα
  - Στην περίπτωσή μας, 2 x (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
- 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 <term, docid> 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.

Παρατήρηση: μπορούμε να εργαστούμε με termid αντί για term αν κρατάμε το λεξικό (την απεικόνιση term, termid) στη μνήμη

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.
- Θα μπορούσαμε να εργαστούμε και με term, docID καταχωρήσεις αντί των termID, docID καταχωρήσεων, αλλά τα ενδιάμεσα αρχεία γίνονται πολύ μεγάλα.

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

Αν 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 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»

- 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
  - Προστίθενται νέοι όροι 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

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 προηγούμενο
  - Κάθε στιγμή, χρησιμοποιούνται κάποια από αυτά
- Έστω n ο αριθμός των postings στη μνήμη
- Διατηρούμε στο δίσκο ευρετήρια I_0, I_1, …
  - I_0 μεγέθους $2^0 * n$, I_1 μεγέθους $2^1 * n$, I_2 μεγέθους $2^2 * n$ …
- 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

```
LMergeAddToken(indexes, Z_0, token)
 1  Z_0 ← Merge(Z_0, {token})
 2  if |Z_0| = n
 3     then for i ← 0 to ∞
 4              do if I_i ∈ indexes
 5                    then Z_{i+1} ← Merge(I_i, Z_i)
 6                          (Z_{i+1} is a temporary index on disk.)
 7                         indexes ← indexes − {I_i}
 8                    else  I_i ← Z_i      (Z_i becomes the permanent index I_i.)
 9                         indexes ← indexes ∪ {I_i}
10                         Break
11              Z_0 ← ∅
```

```
LogarithmicMerge()
1  Z_0 ← ∅      (Z_0 is the in-memory index.)
2  indexes ← ∅
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 $O(T^2)$
- Λογαριθμική συγχώνευση: κάθε καταχώρηση συγχωνεύεται O(log T)φορές, so complexity άρα πολυπλοκότητα O(T log T)

Ερώτημα

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

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

- 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

Μερικοί αριθμοί

- The Indexed Web contains at least 1.71 billion pages (Sunday, 16 March, 2014).
- Each year, Google changes its search algorithm around 500–600 times http://moz.com/google-algorithm-change

Web search engine data centers

- Οι μηχανές αναζήτησης χρησιμοποιούν data centers (Google, Bing, Baidu) κυρίως από commodity μηχανές. *Γιατί; (fault tolerance)*
- Τα κέντρα είναι διάσπαρτα σε όλο τον κόσμο.
- Εκτίμηση: Google ~1 million servers, 3 million processors/cores (Gartner 2007)

http://www.google.com/insidesearch/howsearchworks/thestory/

Θα το δούμε αναλυτικά σε επόμενα μαθήματα
Λίγα «εγκυκλοπαιδικά» για το MapReduce και τη χρήση του στην κατασκευή του ευρετηρίου

Google index

- index partitioned by document IDs into pieces called shards
- each shard is replicated onto multiple servers
- initially, from hard disk drives, now enough servers to keep a copy of the whole index in main memory

- In June 2010, Caffeine continuously crawl and incrementally update the search index
- Index separated into several layers, some updated faster than the others

Μια ματιά στα πολύ μεγάλης κλίμακας ευρετήρια

Παράλληλη κατασκευή

- Maintain a *master* machine directing the indexing job – considered "safe".
- Break up indexing into sets of (parallel) tasks.
- Master machine assigns each task to an idle machine from a pool.

Parallel tasks

- We will use two sets of parallel tasks
  - Parsers
  - Inverters
- Break the input document collection into *splits*
- Each split is a subset of documents (corresponding to blocks in BSBI/SPIMI)

Parsers

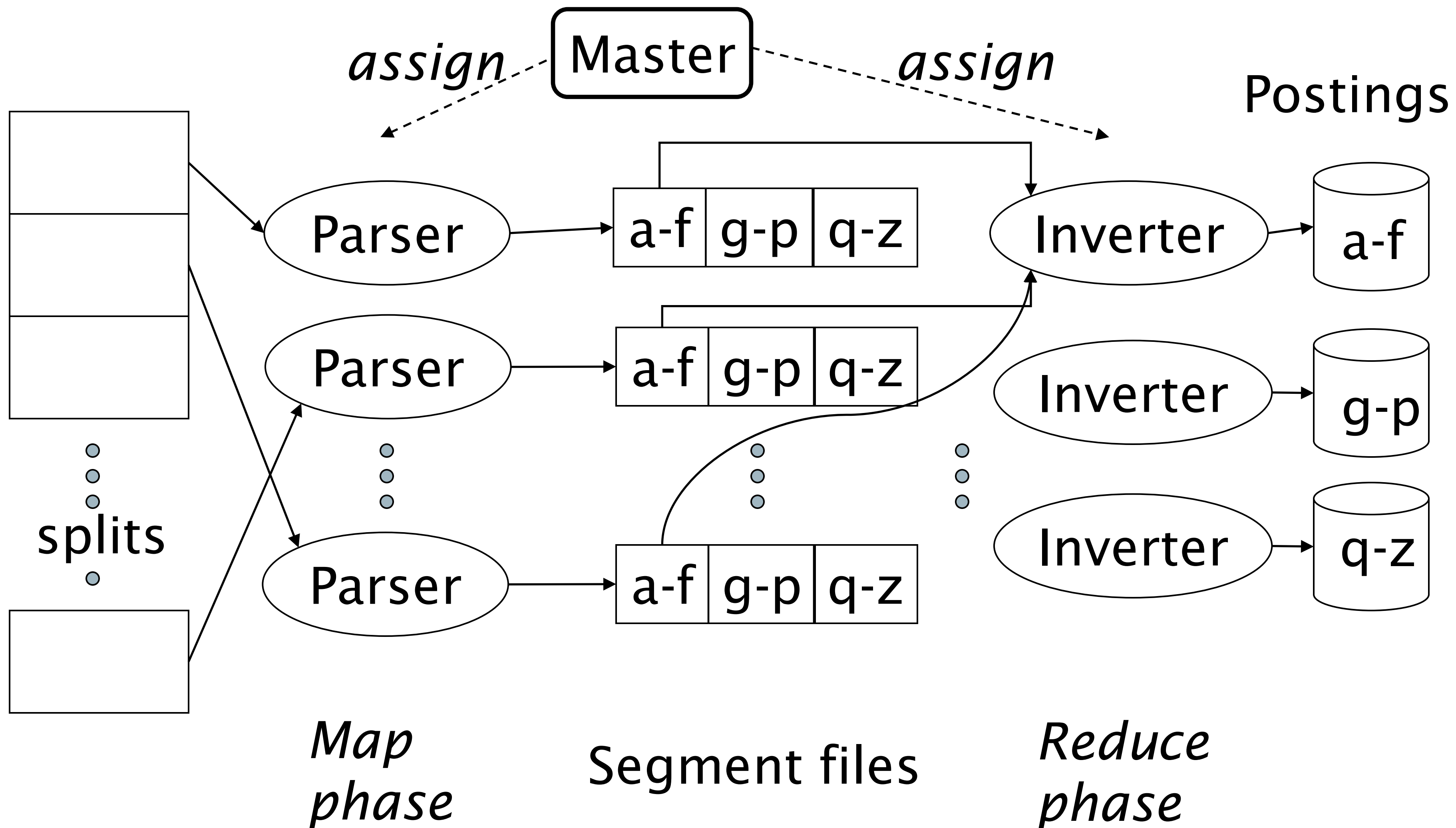
- Master assigns a split to an idle parser machine
- Parser reads a document at a time and emits (term, doc) pairs
- Parser writes pairs into *j* partitions
  - Each partition is for a range of terms' first letters

(e.g., ***a-f, g-p, q-z***) – here *j* = 3.

Inverters

- An inverter collects all (term, doc) pairs (= postings) for one term-partition.
- Sorts and writes to postings lists

Data flow



MapReduce

- The index construction algorithm we just described is an instance of *MapReduce*.
- MapReduce (Dean and Ghemawat 2004) is a robust and conceptually simple framework for distributed computing without having to write code for the distribution part.
- They describe the Google indexing system (ca. 2002) as consisting of a number of phases, each implemented in MapReduce.

*open source implementation as part of Hadoop**

**http://hadoop.apache.org/*

Example for index construction

Map:

- d1 : C came, C c'ed.
- d2 : C died. →

<C,d1>, <came,d1>, <C,d1>, <c'ed, d1>, <C, d2>, <died,d2>

Reduce:

- (<C,(d1,d2,d1)>, <died,(d2)>, <came,(d1)>, <c'ed,(d1)>) →

(<C,(d1:2,d2:1)>, <died,(d2:1)>, <came,(d1:1)>, <c'ed,(d1:1)>)

Schema for index construction in MapReduce

Schema of map and reduce functions

- map: input → list(k, v) reduce: (k, list(v)) → output

Instantiation of the schema for index construction

- map: collection → list(termID, docID)
- reduce: (<termID1, list(docID)>, <termID2, list(docID)>, …) → (postings list1, postings list2, …)

MapReduce

- Index construction was just one phase.
- Another phase: transforming a term-partitioned index into a document-partitioned index.
  - *Term-partitioned:* one machine handles a subrange of terms
  - *Document-partitioned:* one machine handles a subrange of documents
- most search engines use a document-partitioned index … better load balancing, etc.

ΣΤΑΤΙΣΤΙΚΑ ΣΥΛΛΟΓΗΣ

Στατιστικά στοιχεία

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

- Πιο λεπτομερή στατιστικά για τη συλλογή RCV1
  - Πόσο μεγάλο είναι το λεξικό και οι καταχωρήσεις;

Στατιστικά για τη συλλογή Reuters RCV1

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

Μέγεθος ευρετηρίου

size of	word types (terms)			non-positional postings			positional postings		
	dictionary			non-positional index			positional index		
	Size (K)	∆%	cumul %	Size (K)	∆ %	cumul %	Size (K)	∆ %	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

Λεξιλόγιο και μέγεθος συλλογής

- Πόσο μεγάλο είναι το λεξιλόγιο όρων;
  - Δηλαδή, πόσες είναι οι διαφορετικές λέξεις;
- Υπάρχει κάποιο άνω όριο;

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**:

$$M = kT^b$$

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

περιγράφει πως μεγαλώνει το λεξιλόγιο όσο μεγαλώνει η συλλογή

- Συνήθης τιμές: $30 \le k \le 100$ (εξαρτάται από το είδος της συλλογής) και $b \approx 0.5$
- Σε log-log plot του μεγέθους M του λεξιλογίου με το T, ο νόμος προβλέπει γραμμή κλίση περίπου ½

Heaps' Law

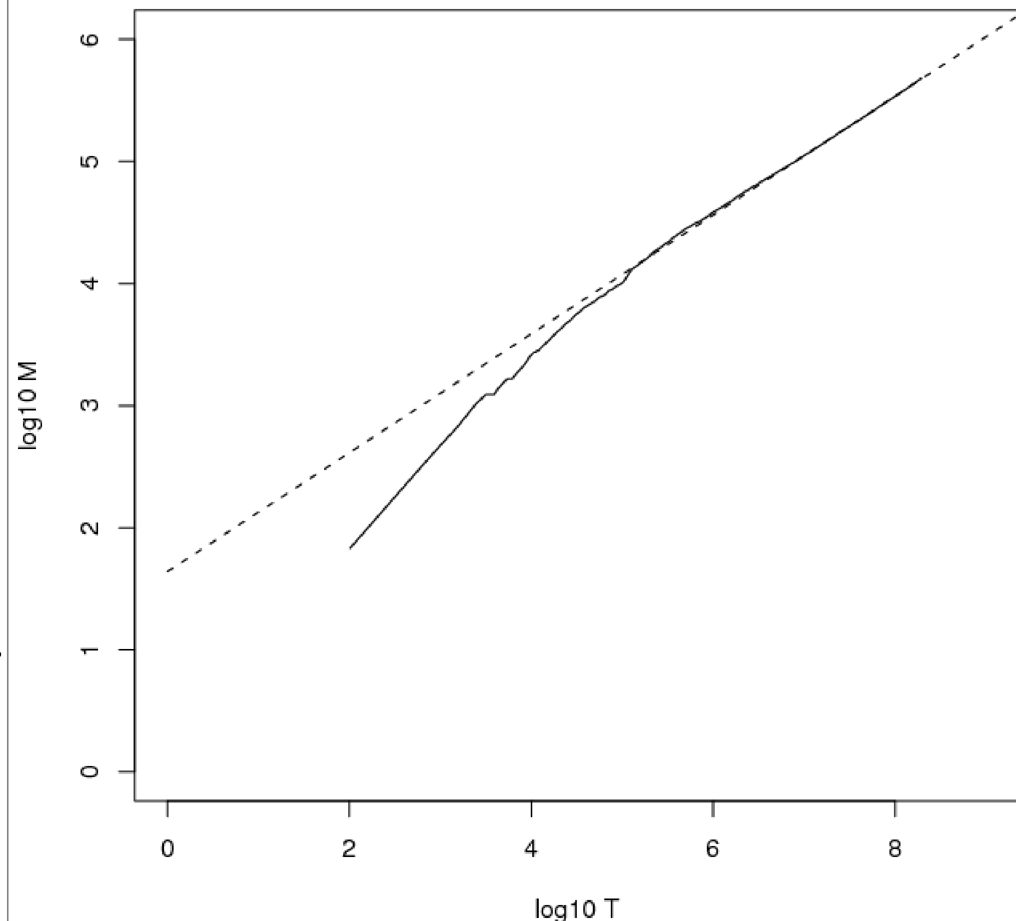
Για το RCV1, η διακεκομμένη γραμμή

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

✓ Ο νόμος του Heaps’ μας δίνει το μέγεθος του λεξιλογίου μιας 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 $1/i$.

$cf_i \propto 1/i = K/i$ όπου K μια normalizing constant

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

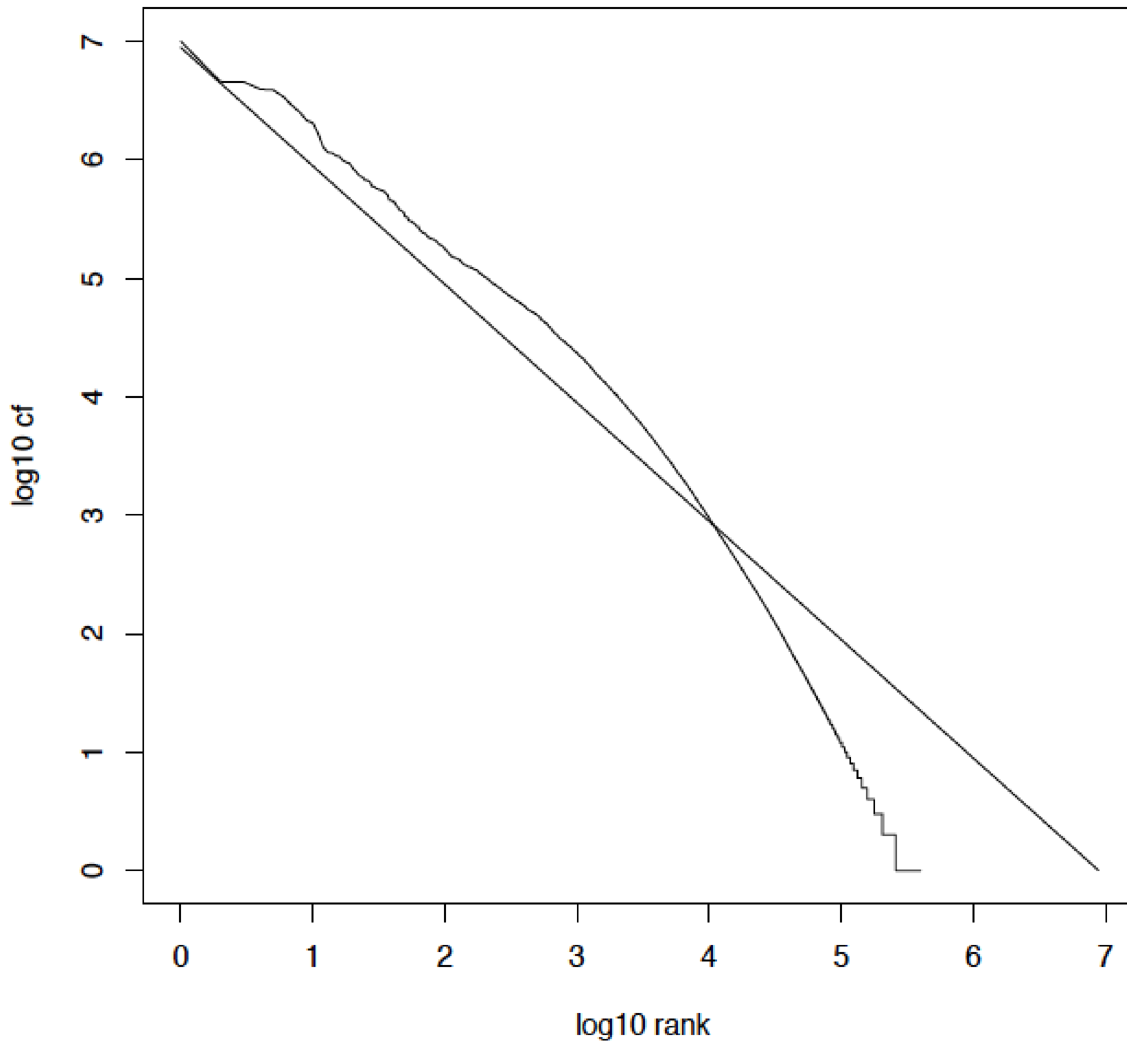
- Αν ο πιο συχνός όρος (ο όρος *the*) εμφανίζεται cf_1 φορές
- Τότε ο δεύτερος πιο συχνός (*of*) εμφανίζεται $cf_1/2$ φορές
- Ο τρίτος (*and*) $cf_1/3$ φορές …

$\log cf_i = \log K - \log i$

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

power law σχέση (εκθετικός νόμος)

Zipf’s law for Reuters RCV1



ΤΕΛΟΣ 5ου Μαθήματος

Ερωτήσεις?

Χρησιμοποιήθηκε κάποιο υλικό των:
- ✓ *Pandu Nayak and Prabhakar Raghavan, CS276: Information Retrieval and Web Search (Stanford)*