Introduction to

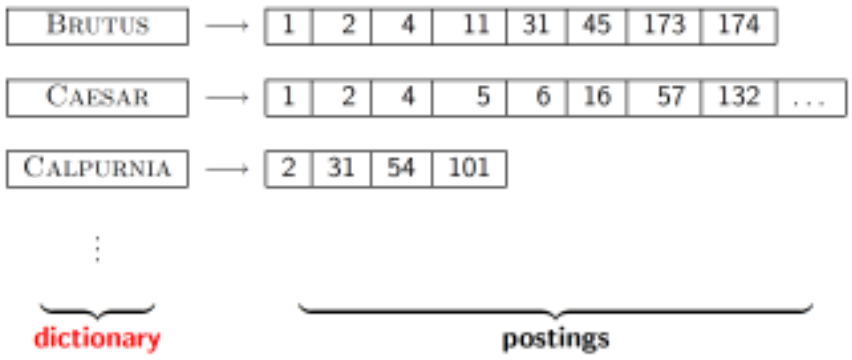
Information Retrieval

ΠΛΕ70: Ανάκτηση Πληροφορίας

Διδάσκουσα: Ευαγγελία Πιτουρά

Διάλεξη 5: Κατασκευή και Συμπίεση Ευρετηρίου

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

Τι θα δούμε σήμερα (1ο μέρος);

- Κατασκευή του Ευρετηρίου σε Μεγάλη Κλίμακα
- Συμπίεση Ευρετηρίου

Υπενθύμιση: 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.

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

Κατασκευή ευρετηρίου

- Πως κατασκευάζουμε το ευρετήριο;
- Ποιες στρατηγικές χρησιμοποιούμε όταν έχουμε περιορισμένη κυρίως μνήμη?
- Εξωτερική διάταξη

Κλιμάκωση της κατασκευής του ευρετηρίου

- Δεν είναι δυνατή η πλήρης κατασκευή του στη μνήμη (in-memory)
  - Δεν μπορούμε να φορτώσουμε όλη τη συλλογή στη μνήμη, να την ταξινομήσουμε και να τη γράψουμε πίσω στο δίσκο
- Πως μπορούμε να κατασκευάσουμε ένα ευρετήριο για μια πολύ μεγάλη συλλογή;
  - Λαμβάνοντας υπ' όψιν τα περιορισμούς και τα χαρακτηριστικά του υλικού. . .

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

- 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

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.

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

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

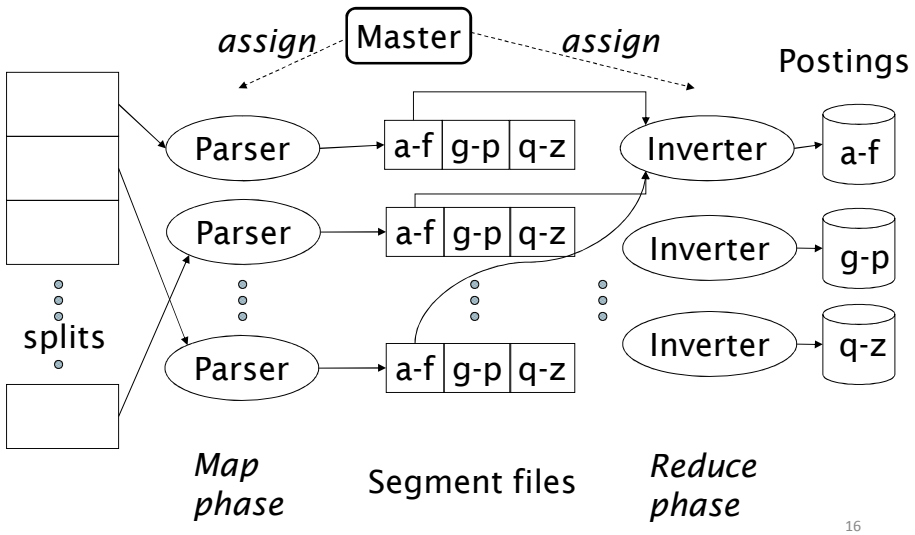
- Master assigns a split to an idle parser machine
- Parser reads a document at a time and emits (term, doc) pairs
- Parser writes pairs into *j* partitions
  - Each partition is for a range of terms' first letters

  (e.g., ***a-f, g-p, q-z***) – here *j* = 3.
- Now to complete the index inversion

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

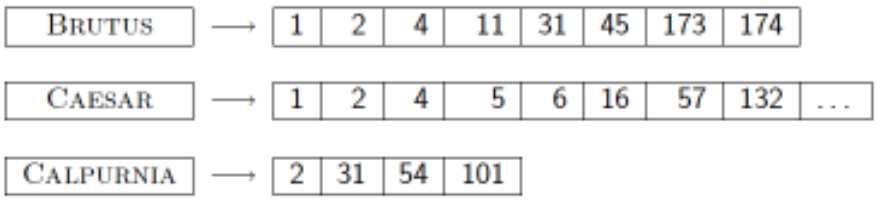
- Index construction was just one phase.
- Another phase: transforming a term-partitioned index into a document-partitioned index.
  - *Term-partitioned:* one machine handles a subrange of terms
  - *Document-partitioned:* one machine handles a subrange of documents
- As we'll discuss in the web part of the course, most search engines use a document-partitioned index … better load balancing, etc.

Introduction to

Information Retrieval

Μερικά θέματα σχετικά με τη συμπίεση

Τι θα δούμε σχετικά με συμπίεση



- Πιο λεπτομερή στατιστικά για τη συλλογή RCV1
  - Πόσο μεγάλο είναι το λεξικό και οι καταχωρήσεις;
- *Συμπίεση του λεξικού*
- *Συμπίεση των καταχωρήσεων*

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

Γιατί συμπίεση των αντεστραμμένων ευρετηρίων;

- Λεξικό
  - Αρκετά μικρό για να το έχουμε στην κύρια μνήμη
  - Ακόμα μικρότερο ώστε να έχουμε επίσης και κάποιες καταχωρήσεις στην κύρια μνήμη
- Αρχείο (α) Καταχωρήσεων
  - Μείωση του χώρου στο δίσκο
  - Μείωση του χρόνου που χρειάζεται για να διαβάσουμε τις λίστες καταχωρήσεων από το δίσκο
  - Οι μεγάλες μηχανές αναζήτησης διατηρούν ένα μεγάλο τμήμα των καταχωρήσεων στη μνήμη

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)	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

Lossless vs. lossy 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 lossy compression
  - Μπορεί να είναι αποδεκτή στην περίπτωση π.χ., που μας ενδιαφέρουν μόνο τα κορυφαία από τα σχετικά έγγραφα

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 πραγματικές 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: $30 \leq k \leq 100$ (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

Για το RCV1, η διακεκομμένη γραμμή

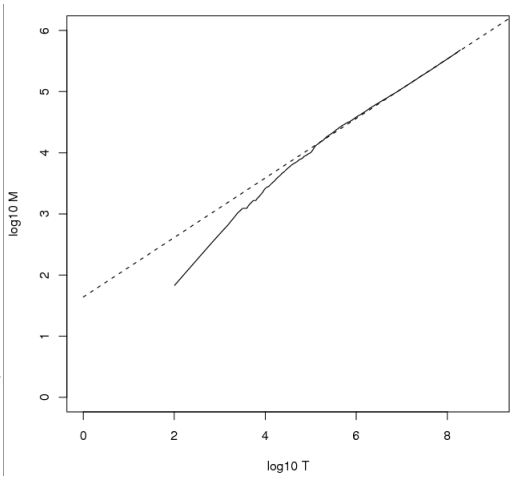
$\log_{10} M = 0.49 \log_{10} T + 1.64$ (το καλύτερο best least squares fit)

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

Heaps' Law



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.

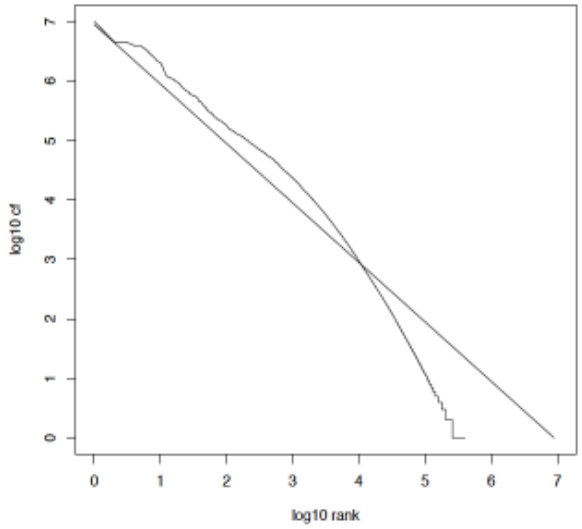
- Αν ο πιο συχνός όρος (ο όρος *the*) εμφανίζεται cf_1 φορές
- Τότε ο δεύτερος πιο συχνός (*of*) εμφανίζεται $cf_1/2$ φορές
- Ο τρίτος (*and*) $cf_1/3$ φορές …

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

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

power law σχέση (εκθετικός νόμος)

Zipf's law for Reuters RCV1



Συμπίεση

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

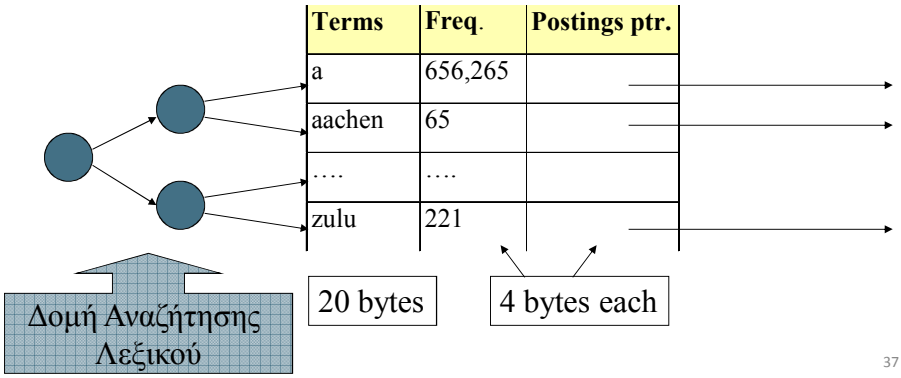
ΣΥΜΠΙΕΣΗ ΛΕΞΙΚΟΥ

Γιατί συμπίεση του λεξικού;

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

Αποθήκευση λεξικού

- Το πιο απλό, ως πίνακα εγγραφών σταθερού μεγέθους (array of fixed-width entries)
  - ~400,000 όροι; 28 bytes/term = 11.2 MB.



Αποθήκευση λεξικού

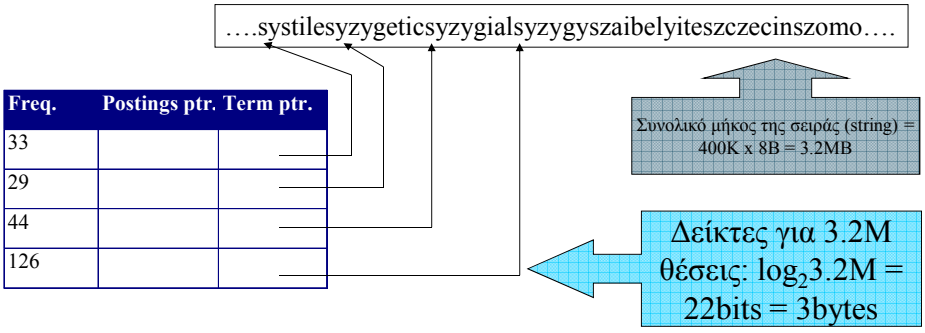
Σπατάλη χώρου

- Πολλά από τα bytes στη στήλη **Term** δε χρησιμοποιούνται – δίνουμε 20 bytes για όρους με 1 γράμμα
- Και δε μπορούμε να χειριστούμε το *supercalifragilisticexpialidocious* ή *hydrochlorofluorocarbons.*
- 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.

Συμπίεση 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.



Χώρος για το λεξικό ως string

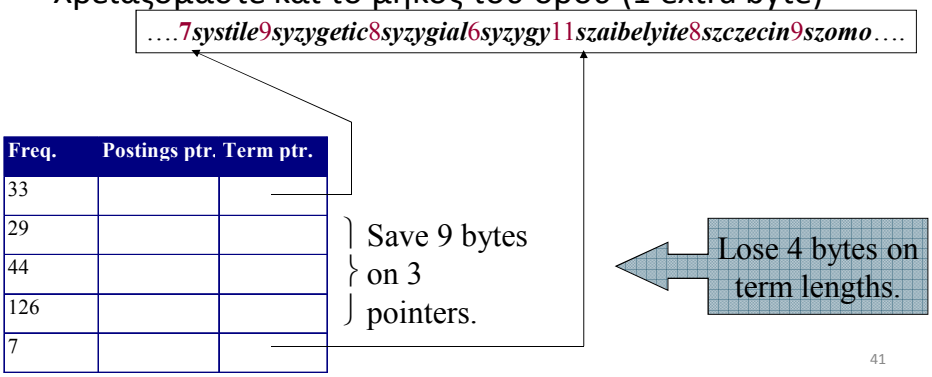
- 4 bytes per term for Freq.
- 4 bytes per term for pointer to Postings.
- 3 bytes per term pointer

} Now avg. 11 bytes/term, not 20.

- Avg. 8 bytes per term in term string
- 400K terms x 19 ⇒ 7.6 MB (against 11.2MB for fixed width)

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)



Blocking

Συνολικό όφελος για block size *k* = 4

- Χωρίς blocking 3 bytes/pointer
  - 3 x 4 = 12 bytes, (ανά block)

Τώρα e 3 + 4 = 7 bytes.

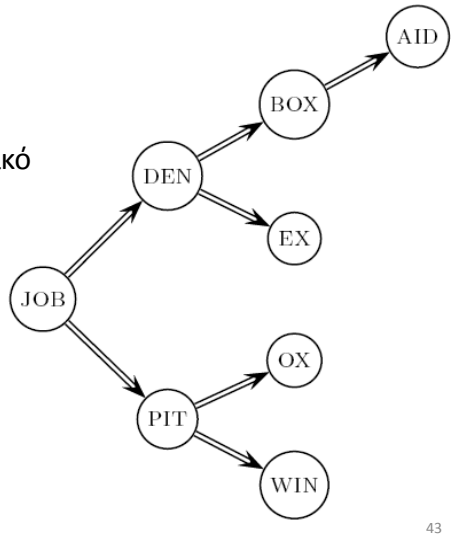
Εξοικονόμηση ακόμα ~0.5MB. Ελάττωση του μεγέθους του ευρετηρίου από 7.6 MB σε 7.1 MB.

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

Αναζήτηση στο λεξικό χωρίς Blocking

- Ας υποθέσουμε δυαδική αναζήτηση και ότι κάθε όρος ισοπίθανο να εμφανιστεί στην ερώτηση (όχι και τόσο ρεαλιστικό στη πράξη) μέσος αριθμός συγκρίσεων = (1+2·2+4·3+4)/8 ~2.6

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



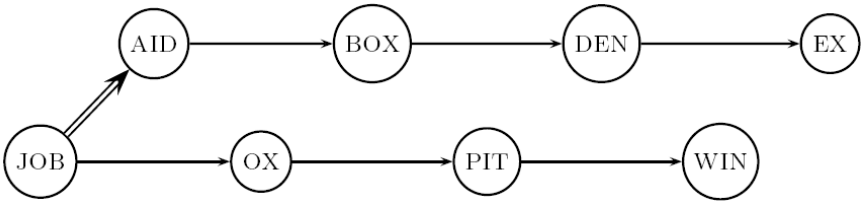
43

Αναζήτηση στο λεξικό με Blocking

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

Μετά γραμμική αναζήτηση στους k = 4 αυτούς όρους.

Μέσος όρος (δυαδικό δέντρο)= (1+2·2+2·3+2·4+5)/8 = 3



Εμπρόσθια κωδικοποίηση (Front coding)

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

8*automata*8*automate*9*automatic*10*automation*

→8*automat***a*1◊*e*2◊*ic*3◊*ion*

Encodes ***automat***

Extra length beyond ***automat.***

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 ≈ 20 bits ανά docID.
- Μπορούμε λιγότερο από 20 bits ανά docID;

Συμπίεση των καταχωρήσεων

- Αποθηκεύουμε τη λίστα των εγγράφων σε αύξουσα διάταξη των docID.
  - ***computer***: 33,47,154,159,202 …
- Συνέπεια: αρκεί να αποθηκεύουμε τα κενά (*gaps*).
  - 33,14,107,5,43 …
- Γιατί; Τα περισσότερα κενά 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								

Συμπίεση 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 ~$\log_2 G$ 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 πετυχαίνουν 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)
  - 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* ≤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 *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.

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

Άλλες κωδικοποιήσεις

- Αντί για bytes, άλλες μονάδες πχ 32 bits (words), 16 bits, 4 bits (nibbles).
- Με byte χάνουμε κάποιο χώρο αν πολύ μικρά διάκενα– nibbles καλύτερα σε αυτές τις περιπτώσεις do better in such cases.
- 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
postings, γ–encoded	101.0

Περίληψη

- 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
  - Η εξοικονόμηση χώρου είναι μικρότερη στην πράξη
  - Αλλά, οι τεχνικές είναι παρόμοιες

ΤΕΛΟΣ 1ου μέρους 5ου Μαθήματος

Ερωτήσεις?

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