Ανάκληση Πληροφορίας

Information Retrieval

Διδάσκων –
Δημήτριος Κατσαρός

Διάλεξη 8η

Το οφέλη της συμπίεσης (γενικώς)

- Χρησιμοποιεί λιγότερο χώρο στον δίσκο
  - Σώζει και κάποια χρήματα
- Διατηρούμε περισσότερα δεδομένα στην κύρια μνήμη
  - Αυξάνει την ταχύτητα
- Αυξάνει την ταχύτητα ανάκτησης δεδομένων από τον δίσκο στην κύρια μνήμη
  - [ανάγνωση συμπιεσμένων δεδομένων & αποσυμπίεση] είναι ταχύτερη από [ανάγνωση ασυμπίεστων δεδομένων]
  - Προϋπόθεση: Οι αλγόριθμοι αποσυμπίεσης είναι ταχείς
    - Αληθές για τους τωρινούς αλγορίθμους

Οφέλη ενός συμπιεσμένου inverted index

- Dictionary
  - Το κάνει μικρό, ώστε να χωράει στην κύρια μνήμη
  - Το κάνει τόσο μικρό, ώστε να χωράνε και κάποιες postings lists (ή τμήματά τους) στην κύρια μνήμη (οι τεχνικές που αναφέρονται ως **static index pruning** δεν θα τις διαπραγματευτούμε στις διαλέξεις, αλλά στις Σειρές Προβλημάτων)
- Postings file(s)
  - Ελάττωση του καταλαμβανόμενου χώρου στον δίσκο
  - Ελάττωση του απαιτούμενου χρόνου για την ανάγνωση των postings lists από τον δίσκο
  - Οι "βιομηχανικές" search engines διατηρούν ένα σημαντικό κομμάτι των postings στην κύρια μνήμη
    - Η Google διατηρεί **ολόκληρο** τον inverted index στην μνήμη! (στην πραγματικότητα, στην μνήμη του cluster)

Θυμηθείτε την συλλογή 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

Παράμετροι του index ως προς το τι κάνουμε indexing

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 ως προς lossy συμπίεση

- Lossless (χωρίς απώλειες) συμπίεση: Όλη η πληροφορία διατηρείται
  - Σχεδόν αποκλειστική τεχνική στην IR
- Lossy (με απώλειες) συμπίεση: Απορρίπτουμε κάποια πληροφορία
- Αρκετά από τα βήματα προεπεξεργασίας που περιγράψαμε σε προηγούμενες διαλέξεις μπορούν να θεωρηθούν ως lossy compression:
  - case folding
  - stop words
  - stemming
  - number elimination (όμως: **Internet Mathematics**, vol. 3, no. 2, pp. 153-185, 2006, "*Inverted index support for numeric search*")

Vocabulary vs. μέγεθος συλλογής

- Πόσο μεγάλο είναι το term vocabulary;
  - Δηλαδή, πόσες διαφορετικές λέξεις υπάρχουν;
- Μπορούμε να υποθέσουμε ένα άνω όριο;
  - Όχι: Τουλάχιστον $70^{20} = 10^{37}$ διαφορετικές λέξεις μεγέθους 20
- Στην πράξη, το vocabulary θα μεγαλώνει διαρκώς καθώς μεγαλώνει το μέγεθος της συλλογής
  - Ειδικά με τους Unicode

Vocabulary vs. μέγεθος συλλογής

- Ο νόμος του Heap: $M = kT^b$
- M είναι το μέγεθος του vocabulary, T είναι ο αριθμός των tokens της συλλογής
- Τυπικές τιμές: $30 \leq k \leq 100$ και $b \approx 0.5$
- Σε log-log διάγραμμα του μεγέθους του vocabulary M ως προς το T, ο νόμος του Heaps προβλέπει μια ευθεία με κλίση περίπου ½
  - Είναι η απλούστερη πιθανή σχέση μεταξή των δυο στον log-log χώρο
  - Είναι εμπειρικός νόμος (“empirical law”)

Ο νόμος του Heaps

Για την RCV1, η στικτή γραμμή

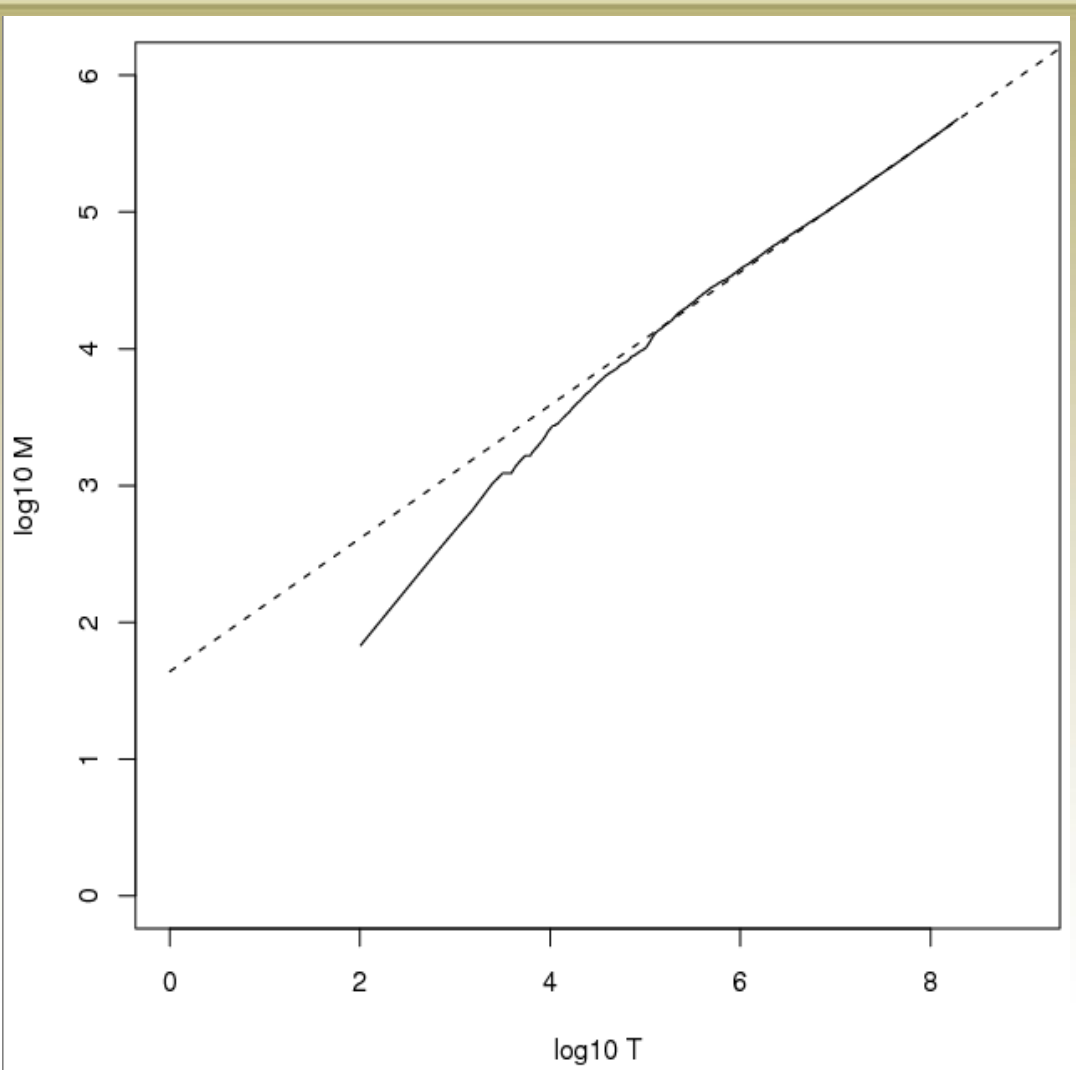
$\log_{10} M = 0.49 \log_{10} T + 1.64$ is είναι το καλύτερο least squares ταίριασμα

Επομένως, $M = 10^{1.64} T^{0.49}$, έτσι $k = 10^{1.64} \approx 44$ και $b = 0.49$

Εξαιρετικό εμπειρικό ταίριασμα για τηνReuters RCV1

Για τα πρώτα 1,000,020 tokens, ο νόμος προβλέπει

38,323 terms; στην πραγματικότητα, τα tokens είναι 38,365



Ο νόμος του Zipf

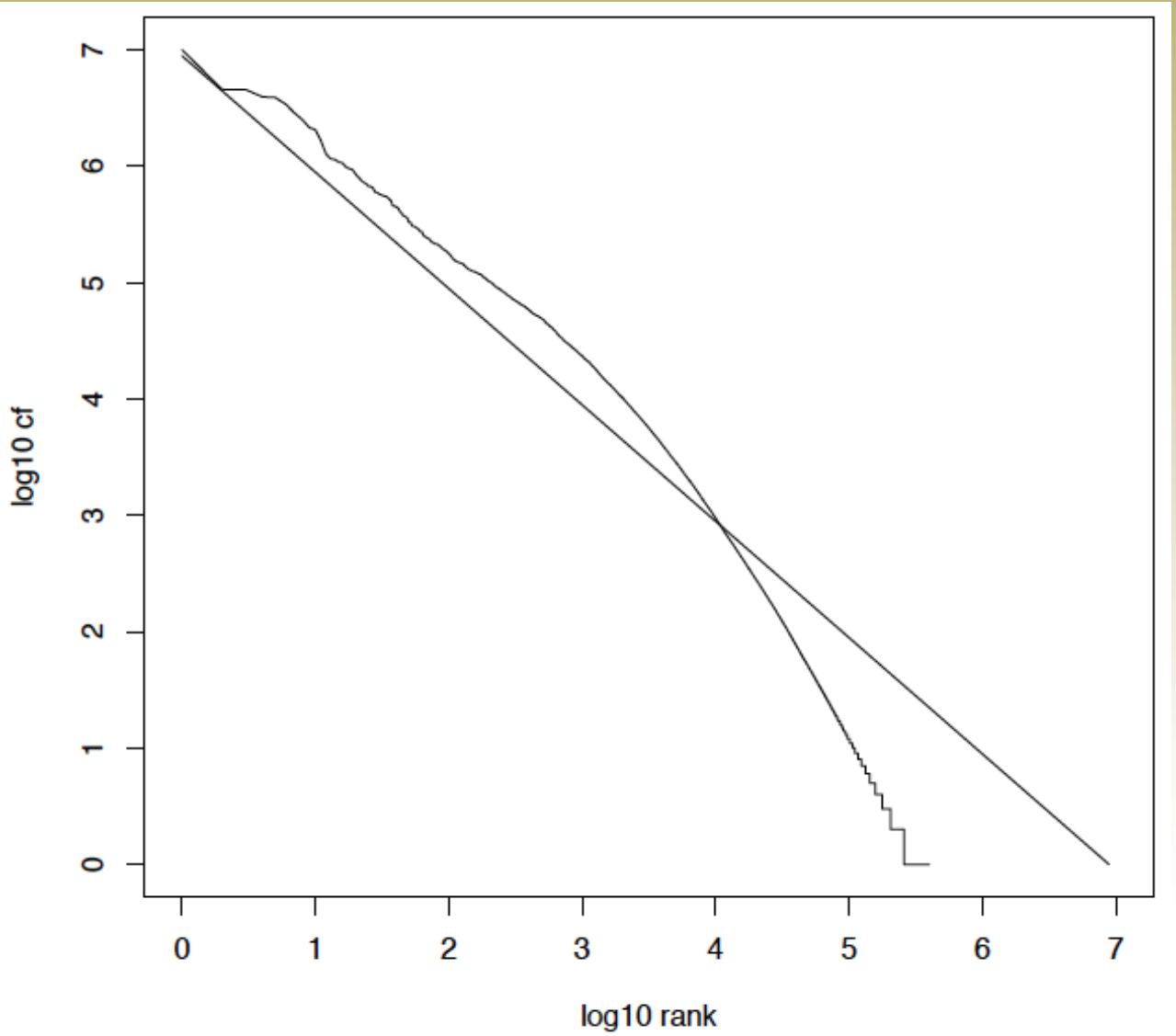
- Ο νόμος του Heaps παρέχει το μέγεθος του vocabulary στις συλλογές εγγράφων
- Μας ενδιαφέρουν οι σχετικές συχνότητες των όρων
- Στην φυσική γλώσσα, υπάρχουν λίγες πολύ συχνές λέξεις, και πάρα πολλές σπάνια χρησιμοποιούμενες
- Νόμος του Zipf: Ο i-οστός πιο συχνός όρος (άρα έχει rank=i) έχει συχνότητα ανάλογης του $1/i$
- $\mathbf{cf_i \propto 1/i \Rightarrow cf_i = K/i}$, όπου K είναι μια σταθερά κανονικοποίησης
  - cf_i είναι η collection frequency: ο αριθμός των εμφανίσεων του όρου t_i στην συλλογή
- **Zipf's law: $r*cf_r = K$** **[δηλ., rank*coll_Freq=constant]**

Συνέπειες του νόμου του Zipf

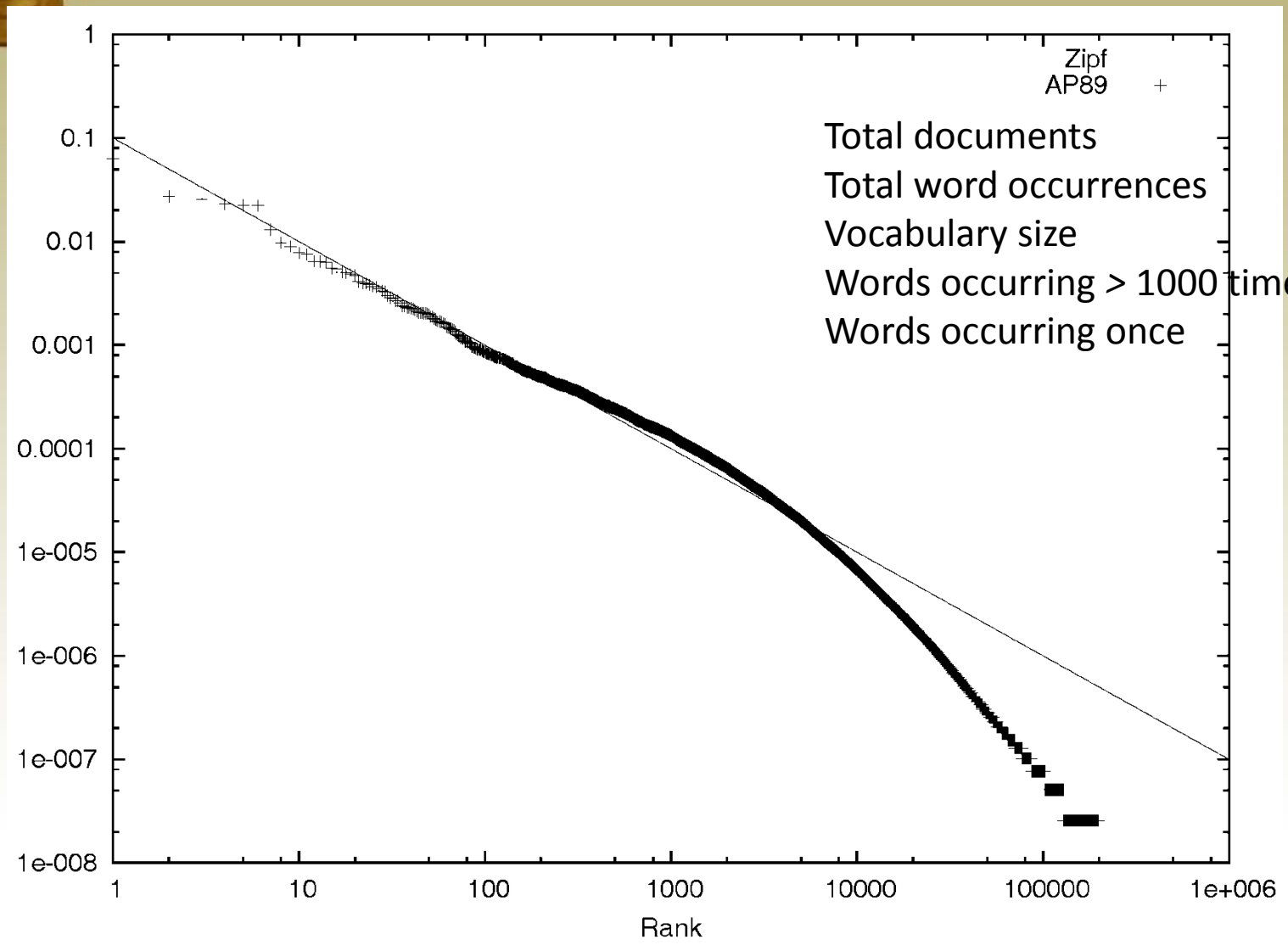
- Εάν ο πιο συχνός όρος (δηλ., *the*) εμφανίζεται cf_1 φορές, τότε
  - ο 2ος πιο συχνός όρος (δηλ., *of*) εμφανίζεται $cf_1/2$ φορές
  - Ο 3ος πιο συχνός όρος (δηλ., *and*) εμφανίζεται $cf_1/3$ φορές
- Από $cf_i * r = K$, προκύπτει: $\log cf_i = \log K - \log r$
  - Γραμμική σχέση μεταξύ $\log cf_i$ και $\log r$
- Μια ακόμη σχέση που διέπεται από κάποιον *δυναμονόμο* (*power law*)

- Εναλλακτική έκφραση του νόμου:
- **Zipf's law: $r*P_r=c$ [δηλ., rank*prob_Occurence=constant]**

Ο νόμος του Zipf στην Reuters RCV1



Zipf στην Associated Press collection '89



Total documents	84,678
Total word occurrences	39,749,179
Vocabulary size	198,763
Words occurring > 1000 times	4,169
Words occurring once	70,064

AP89: Top-50 συχνοί όροι

Word	*Freq.*	*r*	P_r(%)	$r.P_r$	*Word*	*Freq*	*r*	P_r(%)	$r.P_r$
the	2,420,778	1	6.49	0.065	has	136,007	26	0.37	0.095
of	1,045,733	2	2.80	0.056	are	130,322	27	0.35	0.094
to	968,882	3	2.60	0.078	not	127,493	28	0.34	0.096
a	892,429	4	2.39	0.096	who	116,364	29	0.31	0.090
and	865,644	5	2.32	0.120	they	111,024	30	0.30	0.089
in	847,825	6	2.27	0.140	its	111,021	31	0.30	0.092
said	504,593	7	1.35	0.095	had	103,943	32	0.28	0.089
for	363,865	8	0.98	0.078	will	102,949	33	0.28	0.091
that	347,072	9	0.93	0.084	would	99,503	34	0.27	0.091
was	293,027	10	0.79	0.079	about	92,983	35	0.25	0.087
on	291,947	11	0.78	0.086	i	92,005	36	0.25	0.089
he	250,919	12	0.67	0.081	been	88,786	37	0.24	0.088
is	245,843	13	0.65	0.086	this	87,286	38	0.23	0.089
with	223,846	14	0.60	0.084	their	84,638	39	0.23	0.089
at	210,064	15	0.56	0.085	new	83,449	40	0.22	0.090
by	209,586	16	0.56	0.090	or	81,796	41	0.22	0.090
it	195,621	17	0.52	0.089	which	80,385	42	0.22	0.091
from	189,451	18	0.51	0.091	we	80,245	43	0.22	0.093
as	181,714	19	0.49	0.093	more	76,388	44	0.21	0.090
be	157,300	20	0.42	0.084	after	75,165	45	0.20	0.091
were	153,913	21	0.41	0.087	us	72,045	46	0.19	0.089
an	152,576	22	0.41	0.090	percent	71,956	47	0.19	0.091
have	149,749	23	0.40	0.092	up	71,082	48	0.19	0.092
his	142,285	24	0.38	0.092	one	70,266	49	0.19	0.092
but	140,880	25	0.38	0.094	people	68,988	50	0.19	0.093

AP89: Χαμηλής-συχνότητας όροι

Word	*Freq.*	r	$P_r(\%)$	$r.P_r$
assistant	5,095	1,021	.013	0.13
sewers	100	17,110	.000256	0.04
toothbrush	10	51,555	.000025	0.01
hazmat	1	166,945	.000002	0.04

Στατιστικά με βάση των Νόμο του Zipf

- Ποια είναι η αναλογία των όρων με μια δεδομένη συχνότητας εμφάνισης;
  - Όρος που εμφανίζεται n φορές times έχει rank $r_n = k/n$
  - Ο αριθμός όρων με συχνότητα n είναι is
    - $r_n - r_{n+1} = k/n - k/(n+1) = k/n(n+1)$
  - Η αναλογία (δηλ., ποσοστό) βρίσκεται παίροντας το κλάσμα ως προς των συνολικό αριθμό όρων = υψηλότερο rank = k
  - Συνεπώς, η αναλογία με συχνότητα n είναι: $1/n(n+1)$

Παράδειγμα υπολογισμού στατιστικών

- Δειγματικοί όροι frequency ranking

Rank	*Word*	*Frequency*
1000	concern	5,100
1001	spoke	5,100
1002	summit	5,100
1003	bring	5,099
1004	star	5,099
1005	immediate	5,099
1006	chemical	5,099
1007	african	5,098

- Για να υπολογίσουμε τον αριθμό των όρων με συχνότητα εμφάνισης 5,099
  - rank του “chemical” μείον το rank του “summit”
  - 1006 − 1002 = 4

Εκτεταμένο παράδειγμα

Number of Occurrences (n)	*Predicted Proportion (1/n(n+1))*	*Actual Proportion*	*Actual Number of Words*
1	.500	.402	204,357
2	.167	.132	67,082
3	.083	.069	35,083
4	.050	.046	23,271
5	.033	.032	16,332
6	.024	.024	12,421
7	.018	.019	9,766
8	.014	.016	8,200
9	.011	.014	6,907
10	.009	.012	5,893

- Αναλογία όρων που εμφανίζονται *n* φορές σε 336,310 TREC documents
- Μέγεθος vocabulary: 508,209

Συμπίεση

- Τώρα, θα εξετάσουμε τα ζητήματα της συμπίεσης του dictionary και των postings λιστών
  - Μόνο για τον βασικό Boolean index
  - Θα εξετάσουμε και τους positional indexes στις διαλέξεις, αλλά εσείς πρώτα θα μελετήσετε προσεκτικά το άρθρο:
    - L. Akritidis, D. Katsaros, P. Bozanis. "*Improved Retrieval Effectiveness by Efficient Combination of Term Proximity and Zone Scoring: A Simulation-based Evaluation*", **Simulation Modelling: Practice And Theory**, vol. 22, no. 3, pp. 74-91, March, 2012
  - Θα δούμε διάφορα σχήματα συμπίεσης

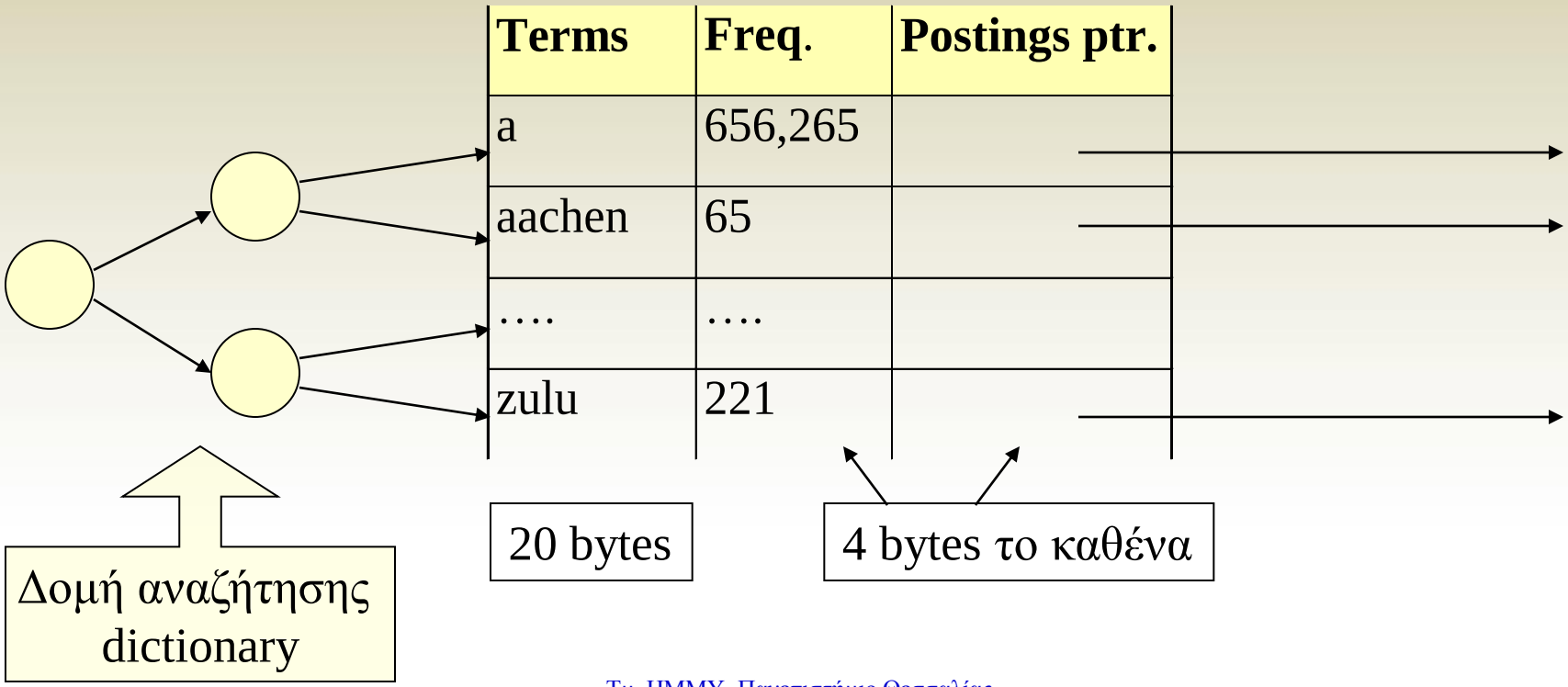
Συμπίεση του Dictionary

Γιατί να συμπιέσουμε το dictionary;

- Η αναζήτηση ξεκινά από το dictionary
- Επιθυμούμε να το αποθηκεύσουμε στην κύρια μνήμη
- Συναγωνίζεται στην χρήση της μνήμης με άλλες εφαρμογές
- Ακόμη και εάν το dictionary δεν είναι αποθηκευμένο στην κύρια μνήμη, επιθυμούμε να είναι μικρό σε μέγεθος για να εκτελείται γρήγορα η αναζήτηση
- Συνεπώς, η συμπίεσή του είναι σημαντικός παράγων

Αποθήκευση του dictionary: Πρώτη προσέγγιση

- Πίνακας κελιών σταθερού πλάτους
  - ~400,000 όροι; 28 bytes/όρο = 11.2 MB

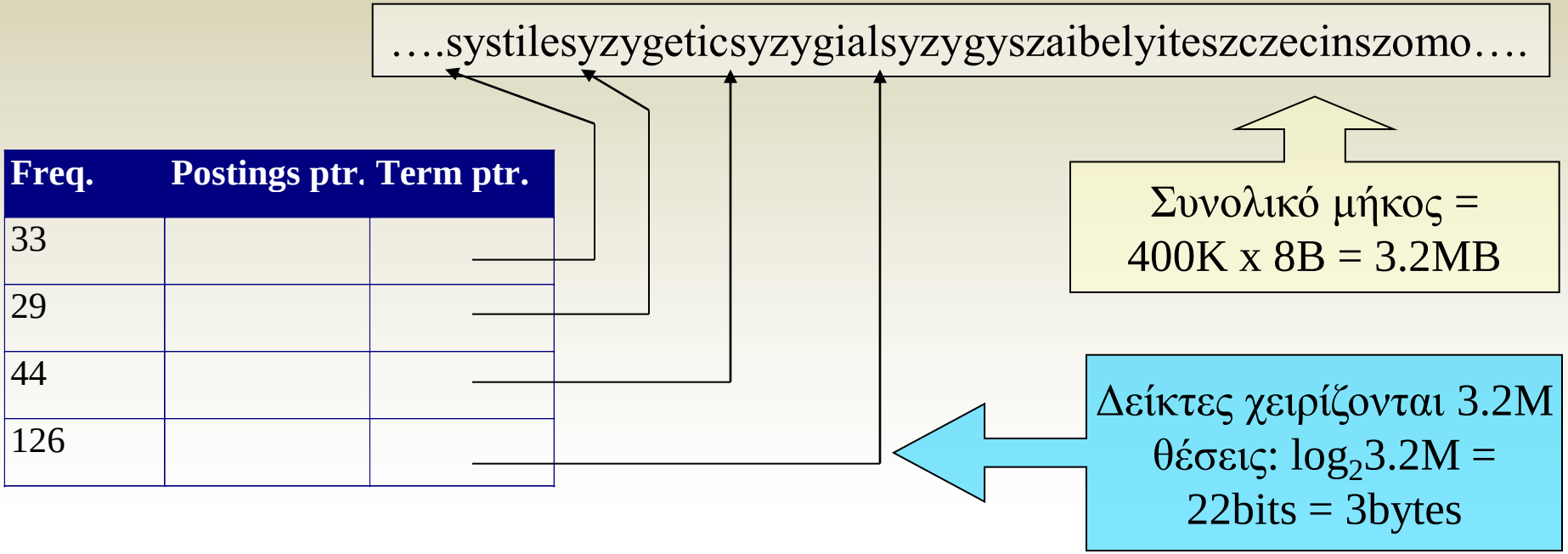


Τα κελιά σταθερού πλάτους σπαταλούν χώρο

- Τα πιο πολλά bytes στην στήλη **Term** σπαταλώνται – δεσμεύουμε 20 bytes για όρους του ενός γράμματος
  - Και φυσικά δεν μπορούμε να χειριστούμε όρους όπως ο *supercalifragilisticexpialidocious* ή ο *hydrochlorofluorocarbons*
- Γραπτά κείμενα στην αγγλική: avg ~4.5 χαρακτήρες/λέξη
- Avg. dictionary λέξη στην αγγλική: ~8 χαρακτήρες
  - Πώς χρησιμοποιούμε ~8 χαρακτήρες ανά όρο του dictionary;
- Οι μικρές λέξεις κυριαρχούν στα tokens

Συμπίεση τις λίστας των όρων: Dictionary-as-a-String

- Αποθήκευση του dictionary ως μια αλυσίδα χαρακτήρων:
  - Δείκτης στην επόμενη λέξη σηματοδοτεί το τέλος της τρεχουσας
  - Ελπίζουμε να επιτύχουμε ελάττωση κατά 60%



Αποθηκευτικό κόστος της τεχνικής dictionary-as-a-string

- 4 bytes ανά όρο για Freq
- 4 bytes ανά όρο για δείκτη σε Postings
- 3 bytes ανά δείκτη όρου

} Πλέον: avg. 11 bytes/όρο, όχι 20

- Avg. 8 bytes ανά όρο στην αλυσίδα όρων
- 400K terms x 19 ⇒ 7.6 MB (σε αντιδιαστολή με τα 11.2MB για πίνακα με σταθερού πλάτους κελιά)

Blocking

- Αποθηκεύουμε δείκτες ανά k-οστή αλυσίδα όρων
  - Το κάτωθι παράδειγμα: k=4
- Χρειάζεται να αποθηκεύσουμε το μήκος των όρων (1 επιπλέον byte)

….7*systile*9*syzygetic*8*syzygial*6*syzygy*11*szaibelyite*8*szczecin*9*szomo*….

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

Κερδίζουμε 9 bytes σε 3 pointers

Πληρώνουμε 4 bytes για τα μήκη των όρων

Blocking

- Παράδειγμα με μέγεθος block size $k = 4$
- Όπου χρησιμοποιήθηκαν 3 bytes/δείκτη χωρίς blocking
  - 3 x 4 = 12 bytes

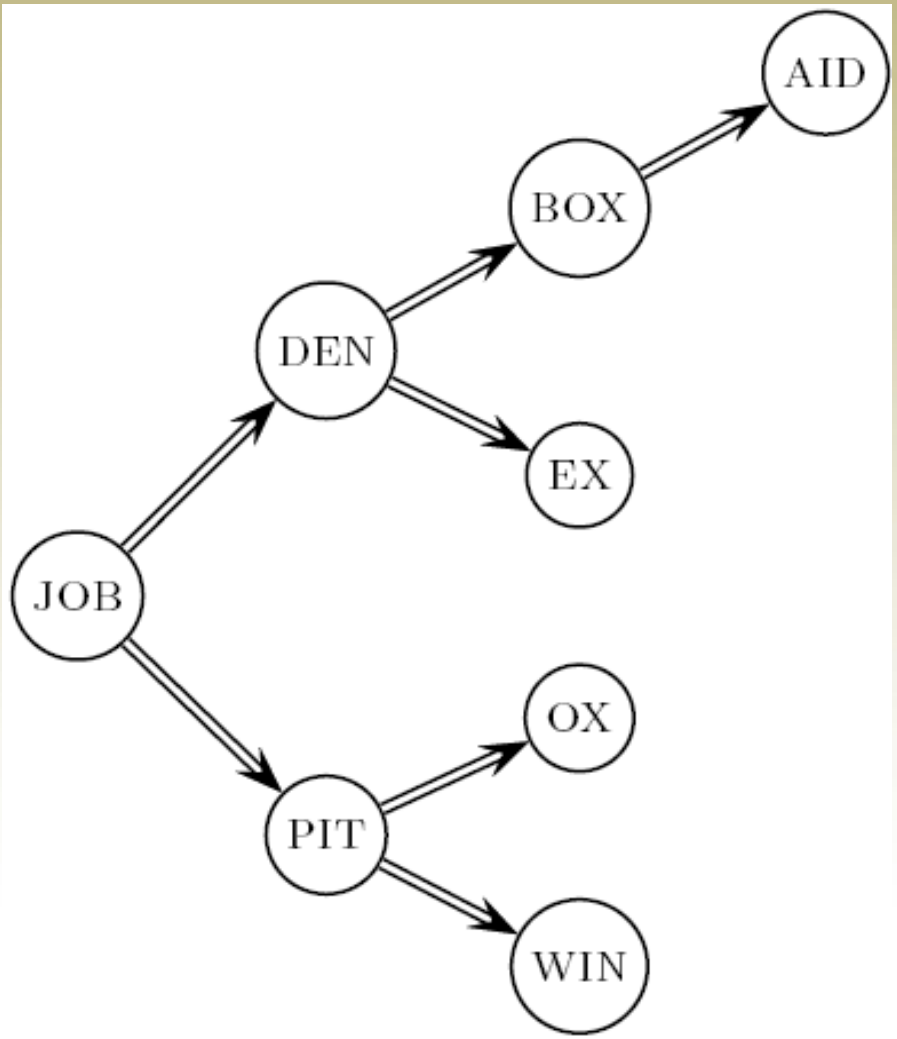
Τώρα χρησιμοποιούνται 3 + 4 = 7 bytes

Ελάττωση κατά ~0.5MB επιπλέον. Αυτό ελαττώνει το μέγεθος του dictionary από τα 7.6 MB στα 7.1 MB. Μπορούμε να επιτύχουμε μεγαλύτερη ελάττωση με μεγαλύτερες τιμές του *k*

Ερώτηση: Γιατί δεν χρησιμοποιούμε μεγαλύτερο *k*;

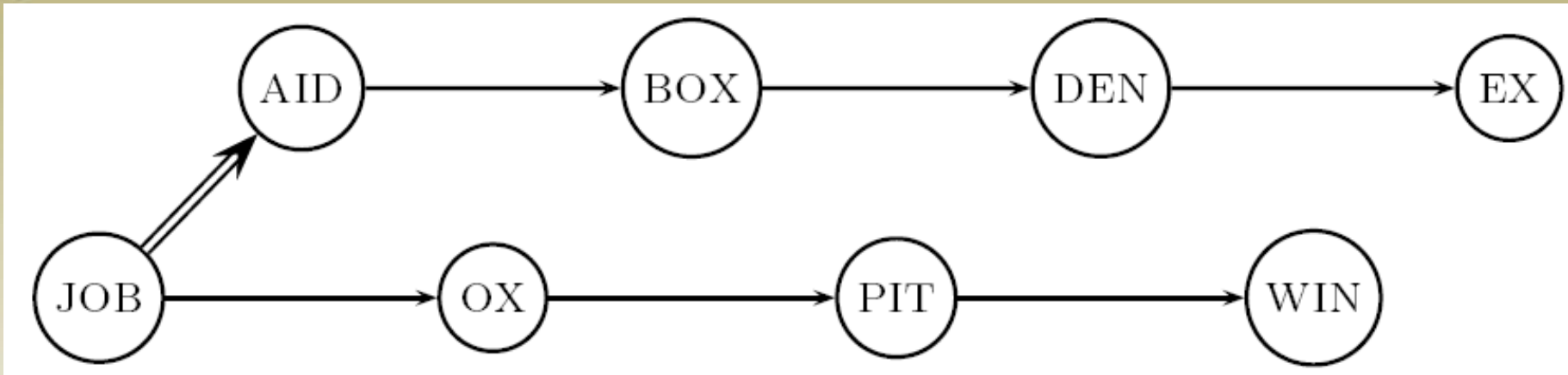
Αναζήτηση στο dictionary χωρίς blocking

- Υποθέτοντας ότι κάθε όρος του dictionary είναι ισοπίθανο να εμφανιστεί στα ερωτήματα (όμως, μη ρεαλιστική υπόθεση!), ο μέσος αριθμός συγκρίσεων είναι:
  (1+2·2+4·3+4)/8=~2.6



Άσκηση: Εάν οι συχνότητες των όρων του ερωτήματος δεν ήταν ομοιόμορφες αλλά ήταν γνωστές, πώς θα διαμορφώνατε το δένδρο αναζήτησης του dictionary;

Αναζήτηση στο dictionary με blocking



- Δυαδική αναζήτηση μέχρι το block των 4 όρων
  - Κατόπιν, σειριακή αναζήτηση ανάμεσα στους όρους του block
- Blocks των 4 (δυαδικό δένδρο), avg=(1+2·2+2·3+2·4+5)/8 = 3 συγκρίσεις

Front coding

- Front-coding:
  - Οι ταξινομημένες λέξεις μοιράζονται συνήθως μακριά, κοινά προθέματα – αποθήκευση μόνο των διαφορών
  - (για τα τελευταία *k-1* σε ένα block των *k*)

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

→8***automat****a*1◊*e*2◊*ic*3◊*ion*

Κωδικοποιεί το ***automat***

Επιπλέον μήκος, πέρα από το ***automat***

Αρχίζει να μοιάζει με την τεχνική συμπίεσης αλφαριθμητικών

RCV1 dictionary compression summary

Τεχνική	Μέγεθος σε MB
Σταθερού μήκους	11.2
Dictionary-as-a-String, με δείκτες σε κάθε όρο	7.6
Με blocking $k = 4$	7.1
Με blocking + front coding	5.9

Συμπίεση των postings

Συμπίεση των postings

- Το postings file είναι κατά πολύ μεγαλύτερο από το dictionary, τουλάχιστον 10 φορές
- Απαίτηση: συμπαγής αποθήκευση κάθε posting
- Ένα posting για τους σκοπούς μας είναι docID
- Για την συλλογή Reuters (800,000 έγγραφα), θα χρειαζόμασταν 32 bits ανά docID, εάν είχαμε ακεραίους των 4-bytes
- Εναλλακτικά, μπορούμε να χρησιμοποιήσουμε $\log_2(800{,}000) \approx 20$ bits ανά docID
- Στόχος μας: χρήση λιγότερων των 20 bits ανά docID

Postings: Δυο αλληλοσυγκρουόμενες δυνάμεις

- Ένας όρος, όπως ***arachnocentric***, εμφανίζεται μια φορά στο ένα εκατομύριο – θα θέλαμε να τον αποθηκεύσουμε κάνοντας χρήση $\log_2(1M)$= ~20 bits
- Ένας όρος, όπως ο ***the***, εμφανίζεται σχεδόν σε κάθε έγγραφο, έτσι τα 20 bits/posting είναι πολύ ακριβό
  - Προτιμούμε 0/1 bitmap vector

Postings file entry

- Αποθηκεύουμε την λίστα των εγγράφων που περιέχουν έναν όρο σε αύξουσα διάταξη του docID
  - ***computer***: 33,47,154,159,202 …
- Συνέπεια: αρκεί να αποθηκεύσουμε *κενά* (*gaps*)
  - 33,14,107,5,43 …
- Ελπίδα: τα περισσότερα κενά μπορεί να κωδικοποιηθούν/αποθηκευτούν με πολύ λιγότερα από 20 bits

Τρεις postings entries

	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								

Κωδικοποίηση μεταβλητού μήκους

- Στόχος:
  - Για τον όρο ***arachnocentric***, θα χρησιμοποιήσουμε ~20 bits/gap
  - Για τον όρο ***the***, θα χρησιμοποιήσουμε ~1 bit/gap
- Εάν το μέσο κενό για έναν όρο είναι G, επιθυμούμε να χρησιμοποιήσουμε **~$\log_2 G$ bits/gap**
- Πρόκληση: κωδικοποίηση κάθε ακεραίου (gap) με όσο το δυνατότερα τα λιγότερα bits που απαιτεί ο συγκεκριμένος ακέραιος
- Αυτό απαιτεί ***variable length encoding***
- Οι κώδικες μεταβλητού μήκους το επιτυγχάνουν με χρήση μικρών κωδικών για τους μικρούς ακεραίους

Κώδικες Variable Byte (VB)

- Για μικρές τιμές κενών G, επιθυμούμε να χρησιμοποιήσουμε ακριβώς τα bytes που απαιτούνται: δηλαδή $\log_2(G)$ bits
- Για να αποθηκεύσουμε το G, αρχίζουμε με ένα byte και αφιερώνουμε 1 bit αυτού ως continuation bit c
- Εάν $G \leq 127$, τον κωδικοποιούμε ως δυαδικό αριθμό στα 7 διαθέσιμα bits και θέτουμε $c=1$
- Αλλιώς, κωδικοποιούμε τα 7 bits χαμηλότερης τάξης του G, και κατόπιν χρησιμοποιούμε επιπλέον bytes για να κωδικοποιούμε τα υψηλής τάξης bits με τον ίδιο τρόπο
- Στο τέλος, θέτουμε το continuation bit του τελευταίου byte ίσο με 1 ($c=1$), και για τα άλλα bytes $c=0$

Παράδειγμα

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

Οι postings αποθηκεύονται ως συνένωση bytes:

000001101011100010000101000011010000110010110001

Ιδιότητα: οι postings κωδικοποιημένες κατά VB έχουν την prefix ιδιότητα

Για μικρό gap (π.χ., 5), η VB κάνει χρήση ολόκληρου byte

Άλλοι κώδικες variable unit

- Αντί για bytes, μπορούμε να χρησιμοποιήσουμε άλλη "μονάδα στοίχισης": 32 bits (words), 16 bits, 4 bits (nibbles)
- Η στοίχιση variable byte σπαταλά χώρο, όταν υπάρχουν πολλά μικρά gaps – τα nibbles αποδίδουν καλύτερα στις περιπτώσεις αυτές
- Οι κώδικες variable byte:
  - Χρησιμοποιούνται από πολλά εμπορικά/ερευνητικά συστήματα
- Υπάρχουν επίσης πρόσφατες εργασίες πάνω σε word-aligned κώδικες που πακετάρουν έναν μεταβλητό αριθμό gaps σε μια word

Μοναδιαίος κώδικας (unary code)

- Αναπαριστούμε τον ακέραιο n ως: n συνεχόμενα 1 και ένα 0 στο τέλος (για να ισχύει η prefix ιδιότητα)
- Ο unary code για το 3 είναι: 1110
- Ο unary code για το 40 είναι:

1111111111111111111111111111111111110 .

- Ο unary code για το 80 είναι:

11111111111111111111111111111111111111111111111111111111111111111111111111110

- Δεν φαίνεται πολύ αποδοτικός κώδικας, αλλά….

γ-codes

- Καλύτερη συμπίεση με bit-level κώδικες
  - Ο γ-κώδικας είναι ο καλύτερος από αυτούς
- Αναπαριστούμε το κενό G ως ζεύγος *length* και *offset*
- *offset*: είναι το G σε δυαδική μορφή, με αποκομμένο το πρώτο bit. Για παράδειγμα: 13 → 1101 → 101
- *length*: είναι το μήκος του offset
  - Για το (offset 101), το μήκος είναι 3
- Κωδικοποιούμε το *length* με *unary code*: 1110
- Ο γ-code του 13 είναι η συνένωση του *length* και του *offset*: 1110101

Παραδείγματα γ-codes

number	length	offset	γ-code
0			none
1	0		0
2	10	0	10,0
3	10	1	10,1
4	110	00	110,00
9	1110	001	1110,001
13	1110	101	1110,101
24	11110	1000	11110,1000
511	111111110	11111111	111111110,11111111
1025	11111111110	0000000001	11111111110,0000000001

Ιδιότητες των γ-codes

- Το G κωδικοποιείται με χρήση: $2\lfloor \log G \rfloor + 1$ bits
  - Το μήκος του offset είναι: $\lfloor \log G \rfloor$ bits
  - Το μήκος του length είναι: $\lfloor \log G \rfloor + 1$ bits
- Όλοι οι γ-codes αποτελούνται από περιττό αριθμό bits
- Σχεδόν δυο φορές χειρότερος από τον βέλτιστο κώδικα ο οποίος έχει μήκος $\log_2 G$
- Ο γ-code έχει την prefix ιδιότητα, όπως και ο VB
- Ο γ-code μπορεί να χρησιμοποιηθεί για κάθε κατανομή ακεραίων
- Ο γ-code δεν απαιτεί παραμετροποίηση

Η γ-συμπίεση χρησιμοποιείται σπάνια

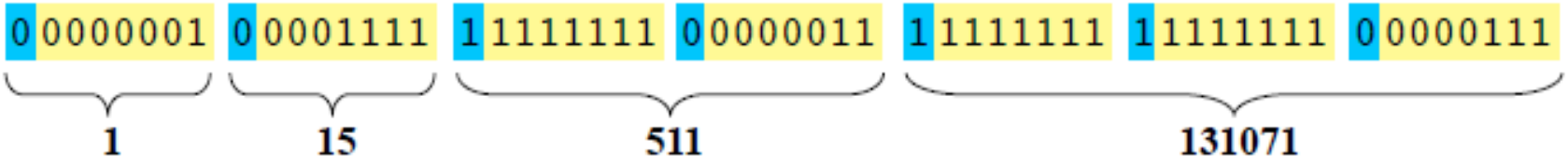
- Οι μηχανές έχουν word boundaries – 8, 16, 32, 64 bits
  - Οι λειτουργίες που “διασχίζουν” αυτά τα όριο είναι πιο αργές
- Η συμπίεση και ο χειρισμός στο επίπεδο των bits μπορεί να είναι αργός
- Η variable byte κωδικοποίηση είναι aligned και συνεπώς εν δυνάμει πιο γρήγορη
- Ανεξάρτητα από την αποδοτικότητα, η variable byte κωδικοποίηση είναι απλούστερη στην σύλληψη, με κόστος λίγο επιπλέον χώρο

Συμπίεση RCV1

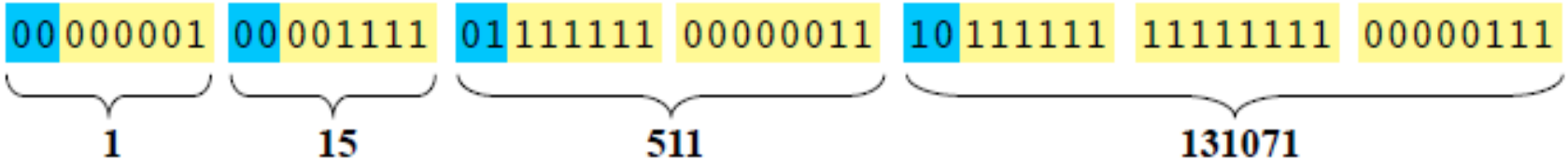
Δομή δεδομένων	Μέγεθος σε MB
dictionary, fixed-width	11.2
dictionary, term pointers into string	7.6
with blocking, k = 4	7.1
with blocking & front coding	5.9
collection (text, xml markup etc)	3,600.0
collection (text)	960.0
Term-doc incidence matrix	40,000.0
postings, uncompressed (32-bit words)	400.0
postings, uncompressed (20 bits)	250.0
postings, variable byte encoded	116.0
postings, γ–encoded	101.0

Μέθοδος συμπίεσης Group VarInt της Google (δημοσιεύτηκε το 2009)

- Varint encoding:
  - 7 bits per byte with continuation bit
  - Con: Decoding requires lots of branches/shifts/masks

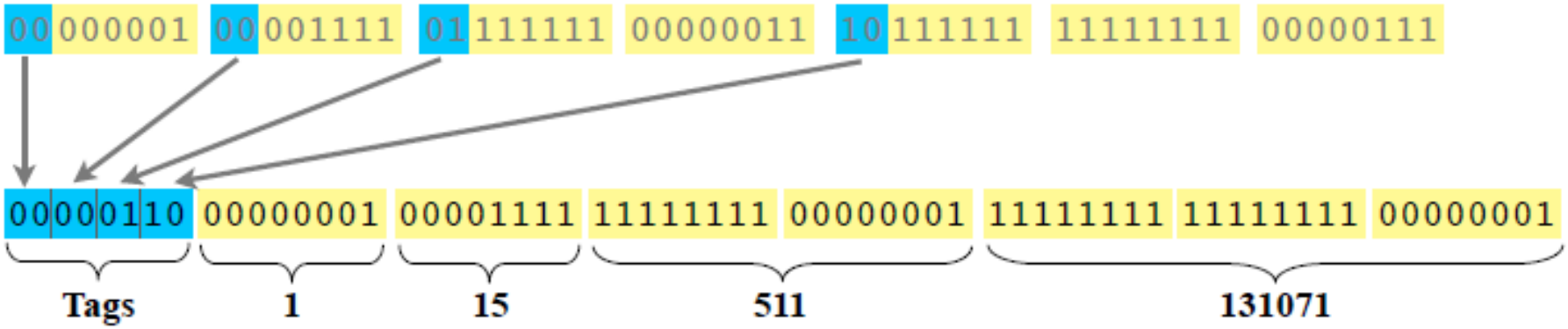


- Idea: Encode byte length as low 2 bits
  - Better: fewer branches, shifts, and masks
  - Con: Limited to 30-bit values, still some shifting to decode



Group Varint κωδικοποιήση

- Idea: encode groups of 4 values in 5-17 bytes
  - Pull out 4 2-bit binary lengths into single byte prefix



- Decode: Load prefix byte and use value to lookup in 256-entry table:

```
...
00|00|01|10  →  Offsets: +1,+2,+3,+5; Masks: ff, ff, ffff, ffffff
...
```

- Much faster than alternatives:
  - 7-bit-per-byte varint: decode ~180M numbers/second
  - 30-bit Varint w/ 2-bit length: decode ~240M numbers/second
  - Group varint: decode ~400M numbers/second

Μήκη κωδικολέξεων & implied distributions

- Κωδικοποίηση του ακεραίου x. Μέγιστος ακέραιος είναι ο m.

Κώδικας	Μήκος κωδικολέξης	Implied distribution
Unary	x+1	$P[x] = 2^{-x}$
Fixed-size binary	$1 + \lfloor \log_2(m) \rfloor$	$P[x] = \frac{1}{m}$
Elias γ-code	$2 \times \lfloor \log_2(x) \rfloor + 1$	$P[x] \approx \frac{1}{2x^2}$
Elias δ-code	$1 + log_2(x) + 2log_2[log_2(x)]$	$P[x] \approx 1/2x[log(x)]^2$
Rice code παρουσίαση στην τάξη	$\left\lfloor \frac{x-1}{2^k} \right\rfloor + 1 + k$	$P[x] = p(1-p)^{x-1}$
PForDelta παρουσίαση στην τάξη	b or b+w	Gaussian
Variable-byte	$\left\lceil \frac{\lvert B(x) \rvert}{7} \right\rceil$	$P[x] \approx \sqrt[7]{1/x}$