Introduction to

Information Retrieval

ΠΛΕ70: Ανάκτηση Πληροφορίας

Διδάσκουσα: Ευαγγελία Πιτουρά

Διάλεξη 6: Συμπίεση Ευρετηρίου

Τι είδαμε στο προηγούμενο μάθημα

- Κατασκευή ευρετηρίου
- Στατιστικά στοιχεία (νόμοι των Heaps και Zipf)

Κατασκευή ευρετηρίου

Είσοδος: N έγγραφα

Έξοδος: Λεξικό και Αντεστραμμένο ευρετήριο

Επεξεργαζόμαστε τα έγγραφα για να βρούμε τις λέξεις (όρους) - αυτές αποθηκεύονται μαζί με το Document ID.

Τι συμβαίνει όταν δεν είναι δυνατή η πλήρης κατασκευή του λεξικού και κυρίως του αντεστραμμένο ευρετήριο στη μνήμη;

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

Στη συνέχεια, για κάθε όρο, διάταξη εγγράφων

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

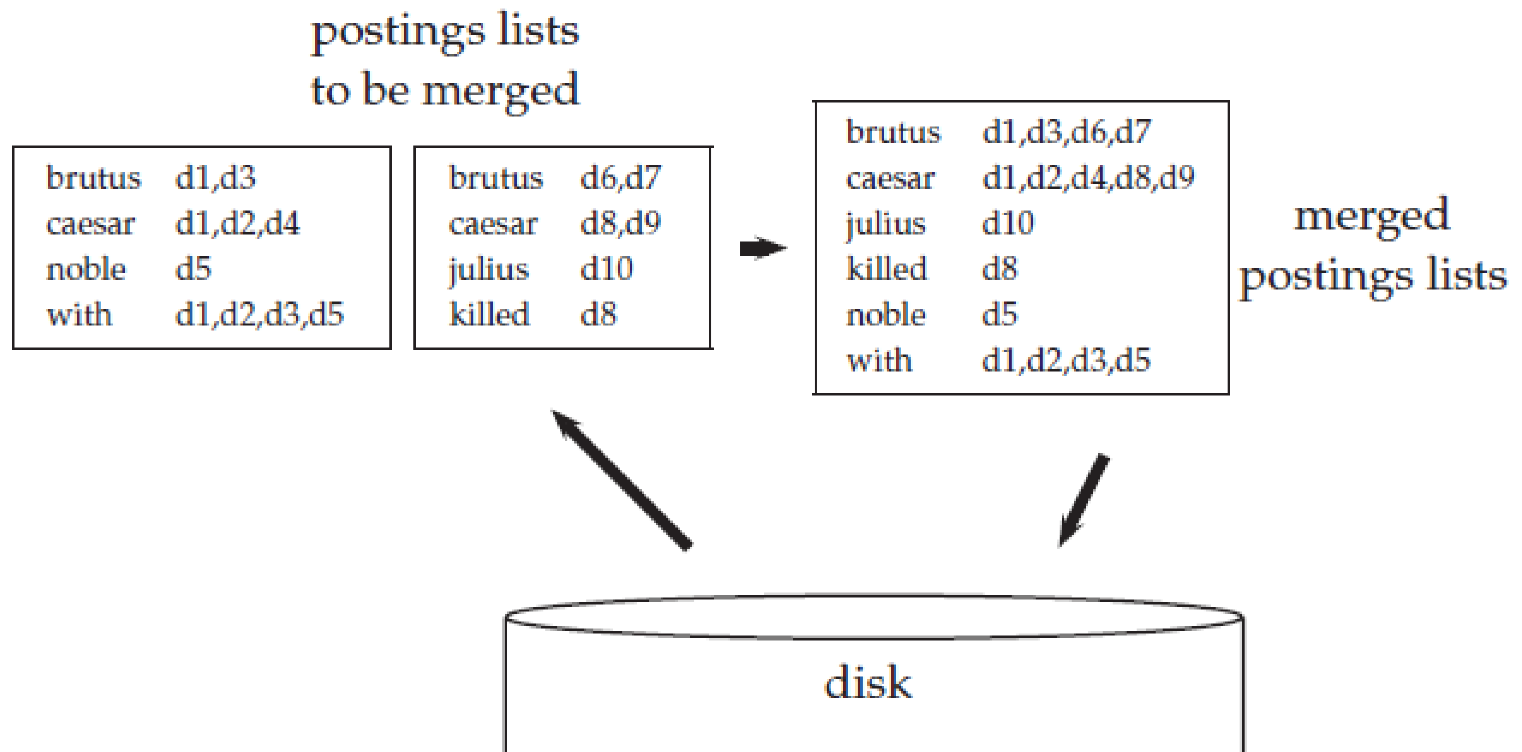
→

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

BSBI: Αλγόριθμος κατασκευής ανά block

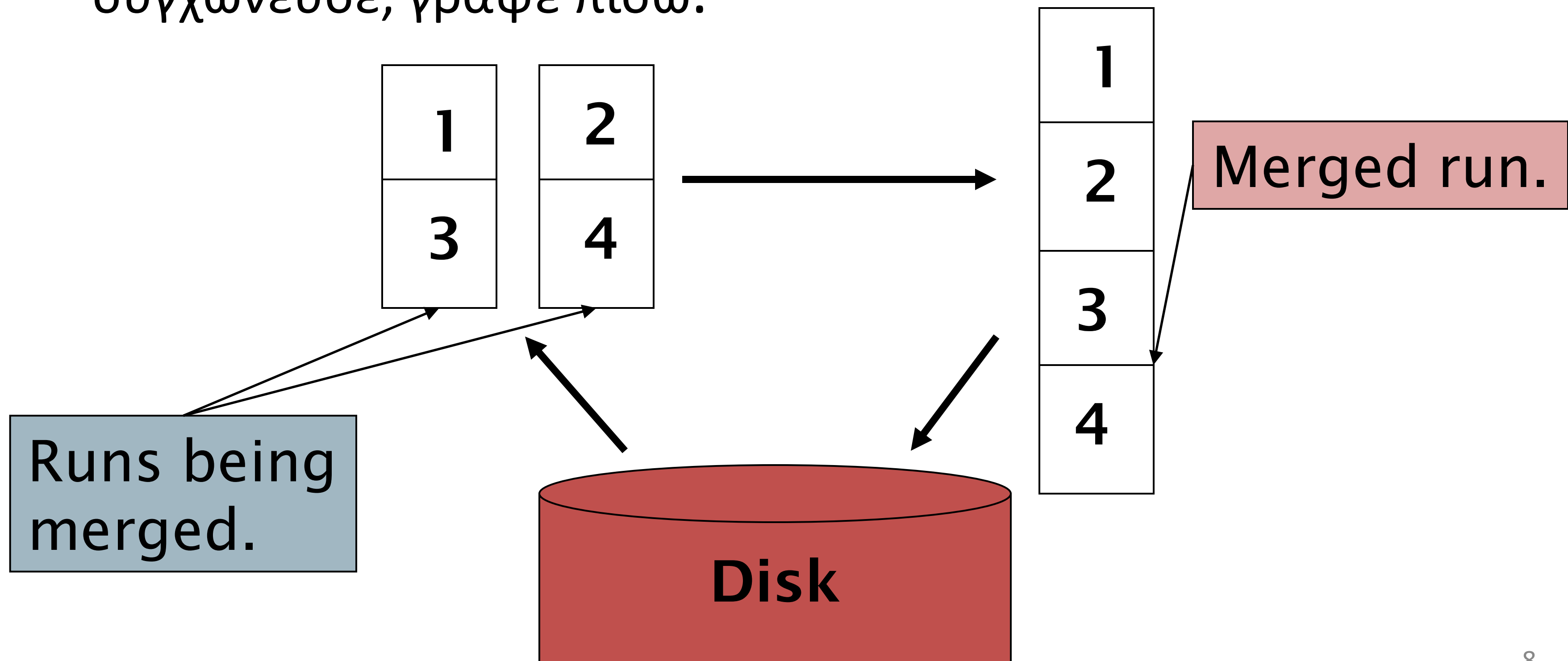
1. Χώρισε τη συλλογή σε κομμάτια ίσου μεγέθους, ώστε κάθε κομμάτι να χωρά στη μνήμη
2. Ταξινόμησε τα ζεύγη termID–docID για κάθε κομμάτι στη μνήμη
3. Αποθήκευσε τα ενδιάμεσα αποτελέσματα (runs) στο δίσκο
4. Συγχώνευσε τα ενδιάμεσα αποτελέσματα

Παράδειγμα



Πως θα γίνει η συγχώνευση των runs?

- Δυαδική συγχώνευση, μια δεντρική δομή, π.χ., για m = 10 runs, $\log_2 10 = 4$ επίπεδα.
- Σε κάθε επίπεδο, διάβασε στη μνήμη runs σε blocks, 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

Δυναμικά ευρετήρια

- Μέχρι στιγμής, θεωρήσαμε ότι τα ευρετήρια είναι στατικά (δηλαδή, δεν αλλάζουν).
- 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

Πολυπλοκότητα

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)

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, Z0, token)
 1  Z0 ← Merge(Z0, {token})
 2  if |Z0| = n
 3    then for i ← 0 to ∞
 4           do if Ii ∈ indexes
 5                then Zi+1 ← Merge(Ii, Zi)
 6                      (Zi+1 is a temporary index on disk.)
 7                     indexes ← indexes − {Ii}
 8                else Ii ← Zi    (Zi becomes the permanent index Ii.)
 9                     indexes ← indexes ∪ {Ii}
10                     Break
11         Z0 ← ∅
```

```
LogarithmicMerge()
1  Z0 ← ∅    (Z0 is the in-memory index.)
2  indexes ← ∅
3  while true
4  do LMergeAddToken(indexes, Z0, 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, άρα πολυπλοκότητα O(T log T)

Ερώτημα

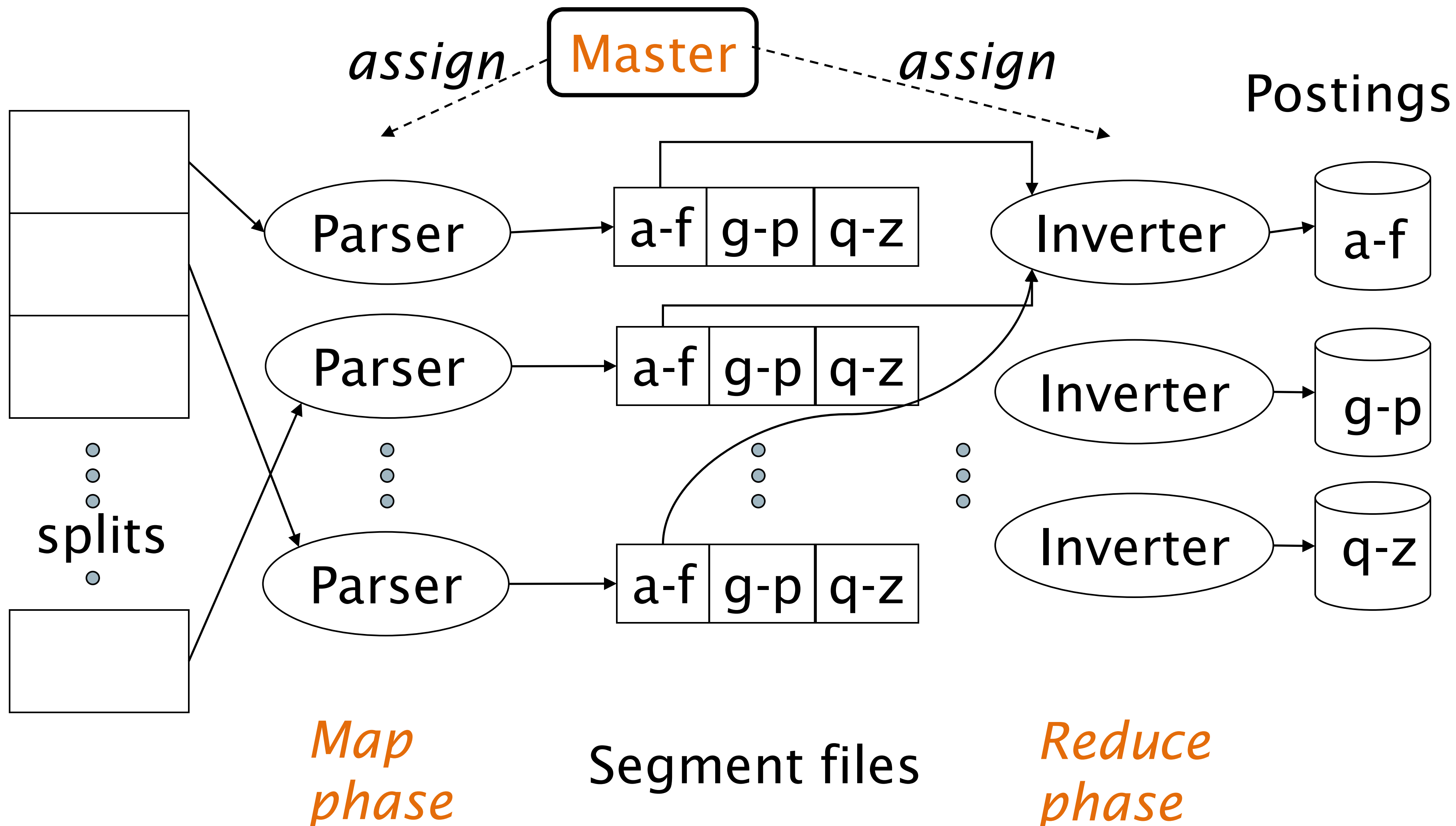
- *Κυρίως και βοηθητικό ευρετήριο*: O(1)
- *Λογαριθμική συγχώνευση*: κοιτάμε O(log T) ευρετήρια

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

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
  - Εκτίμηση: Google ~1 million servers, 3 million processors/cores (Gartner 2007)

Παράλληλη κατασκευή



Παράδειγμα κατασκευής ευρετηρίου σε MapReduce

Το γενικό σχήμα των συναρτήσεων map και reduce

- map: input → list(***key***, value)
- reduce: (***key***, list(value)) → output

Εφαρμογή στην περίπτωση της κατασκευής ευρετηρίου

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

Το ευρετήριο της Google

- Το ευρετήριο κατανέμεται με βάση τα doc IDs σε τμήματα που καλούνται shards
- Υπάρχουν αντίγραφα για κάθε shard σε πολλούς servers
- Σήμερα, όλο το ευρετήριο στη μνήμη

- Από τον Ιουνίου του 2010, Caffeine: συνεχές crawl και σταδιακή ενημέρωση του ευρετηρίου
- Το ευρετήριο χωρίζεται σε επίπεδα με διαφορετική συχνότητα ανανέωσης

Τι είδαμε στο προηγούμενο μάθημα

- Κατασκευή ευρετηρίου
- Στατιστικά στοιχεία (νόμοι των Heaps και Zipf)

Λεξιλόγιο και μέγεθος συλλογής

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

✓ Το λεξιλόγιο συνεχίζει να μεγαλώνει με το μέγεθος της συλλογής

  ✓ Πως? Με βάση το **νόμο του Heaps**

Νόμος του Heaps

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

Heaps' Law

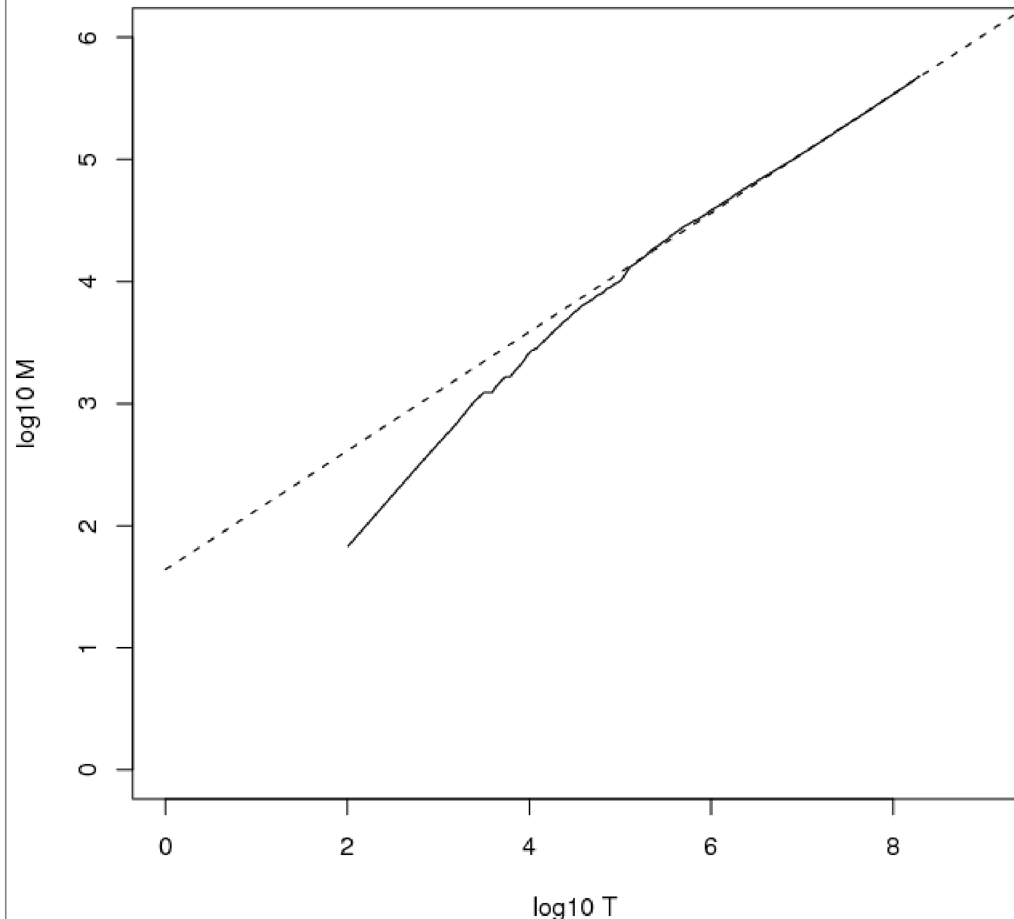
Για το RCV1, η διακεκομμένη γραμμή

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

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

Θα εξετάσουμε τη σχετική συχνότητα των όρων

- Στις φυσικές γλώσσες, υπάρχουν λίγοι όροι που εμφανίζονται πολύ συχνά και πάρα πολύ όροι που εμφανίζονται σπάνια

Ο νόμος του Zipf

Ο νόμος του Zipf:

Ο i-οστός πιο συχνός όρος έχει συχνότητα (collection frequency) cf_i ανάλογη του 1/*i*.

$$cf_i \propto K / i$$

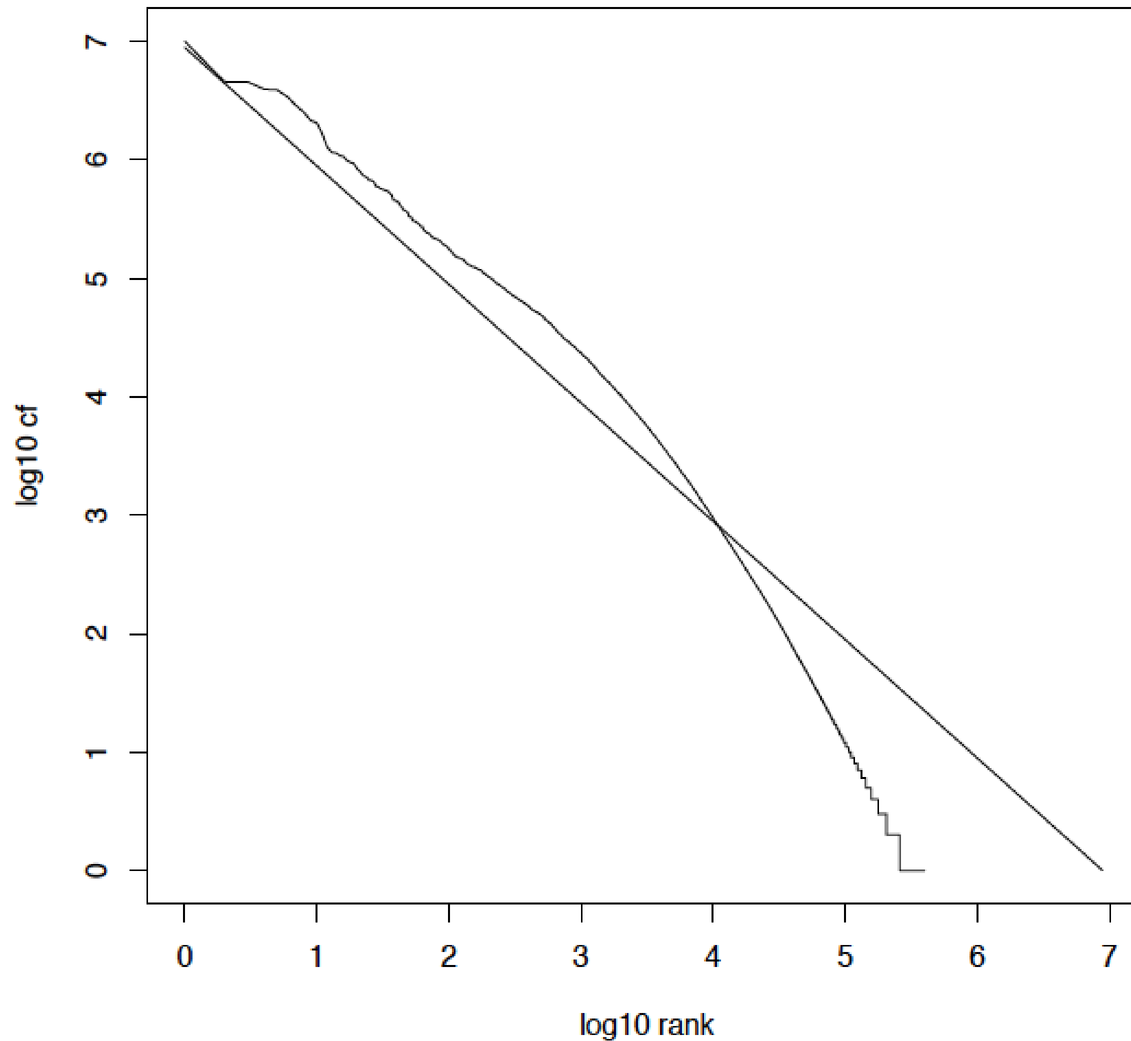
- Αν ο πιο συχνός όρος (ο όρος *the*) εμφανίζεται cf_1 φορές
- Τότε ο δεύτερος πιο συχνός (*of*) εμφανίζεται $cf_1/2$ φορές
- Ο τρίτος (*and*) $cf_1/3$ φορές …

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

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

power law σχέση (εκθετικός νόμος)

Zipf’s law for Reuters RCV1



Τι θα δούμε σήμερα

- Συμπίεση Ευρετηρίου

Συμπίεση

- Θα δούμε μερικά θέματα για τη συμπίεση το λεξιλογίου και των 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

Συμπίεση

- 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

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

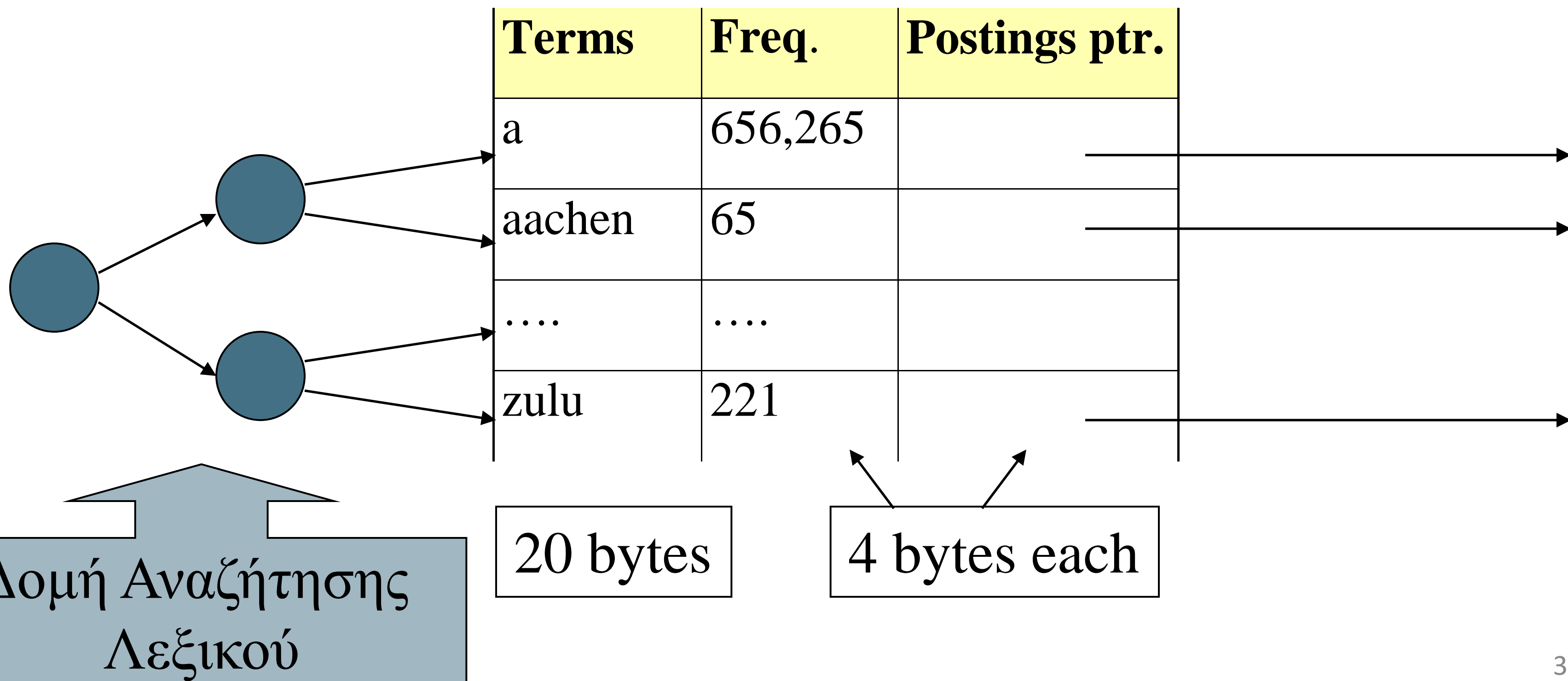
ΣΥΜΠΙΕΣΗ ΛΕΞΙΚΟΥ

Συμπίεση 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

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



Terms	Freq.	Postings ptr.
a	656,265	
aachen	65	
….	….	
zulu	221	

Αποθήκευση λεξικού

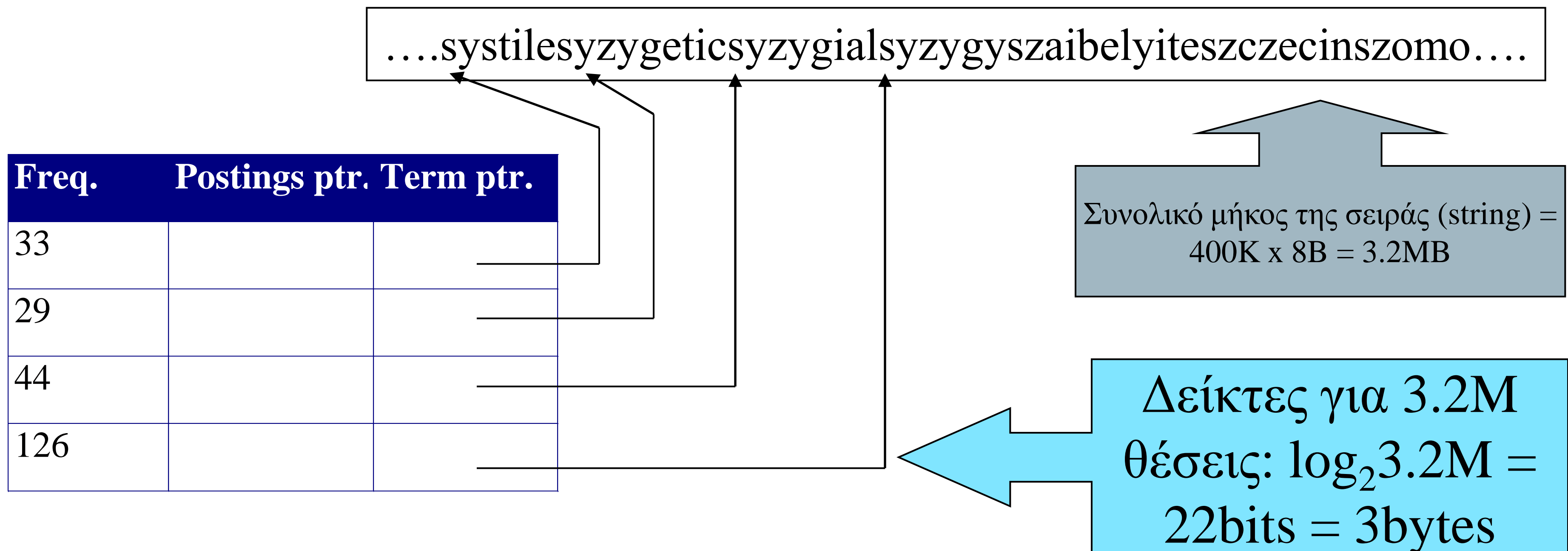
Σπατάλη χώρου

- 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
- Avg. 8 bytes per term in term string (3.2MB)
- 400K terms x 19 ⇒ 7.6 MB (against 11.2MB for fixed width)

} Now avg. 11 bytes/term

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)

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

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

Save 9 bytes on 3 pointers.

Lose 4 bytes on term lengths.

Blocking

Συνολικό όφελος για block size *k* = 4

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

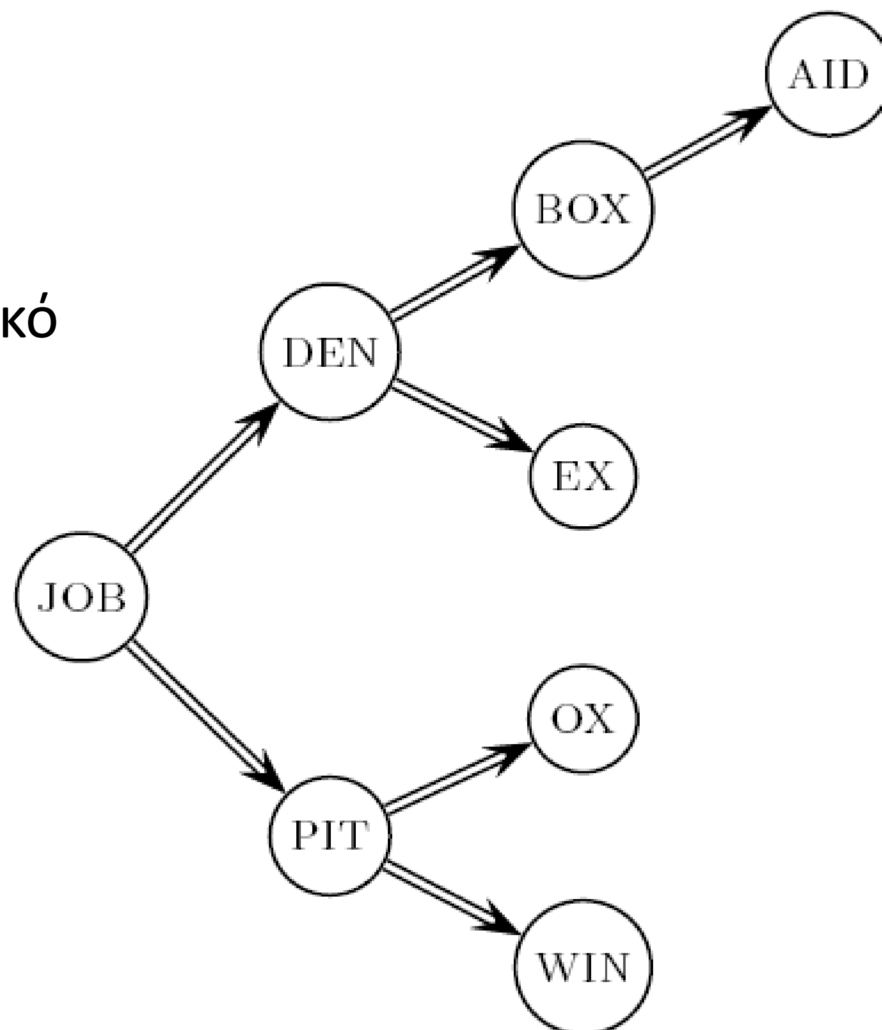
Τώρα 3 + 4 = 7 bytes.

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



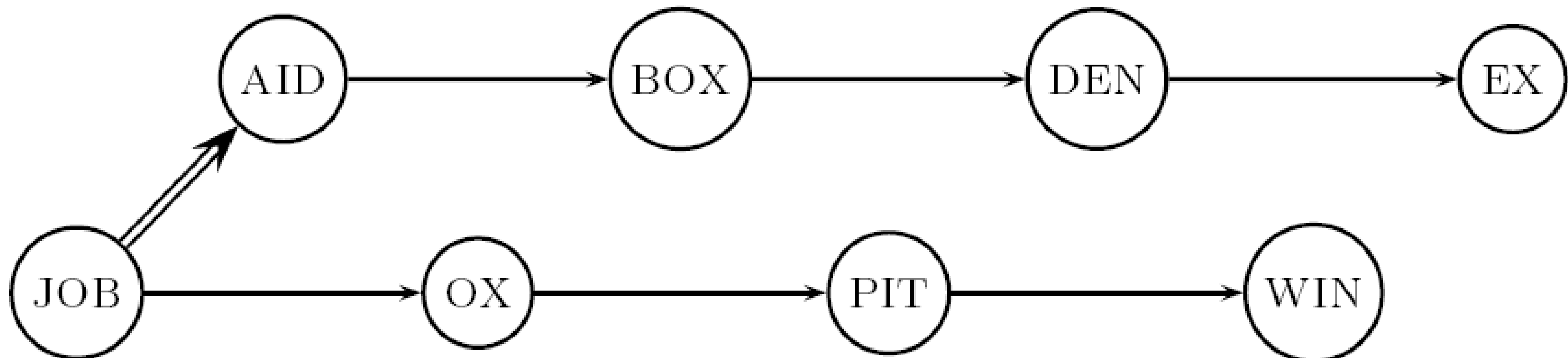
Άσκηση: σκεφτείτε ένα καλύτερο τρόπο αναζήτησης αν δεν έχουμε ομοιόμορφη κατανομή των όρων

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****a1◊***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, $\log_2 800{,}000 \approx 20$ 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 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)
  - 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 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.

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.
- 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 6ου Μαθήματος

Ερωτήσεις?

Χρησιμοποιήθηκε κάποιο υλικό των:

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