Introduction to

Information Retrieval

ΠΛΕ70: Ανάκτηση Πληροφορίας

Διδάσκουσα: Ευαγγελία Πιτουρά

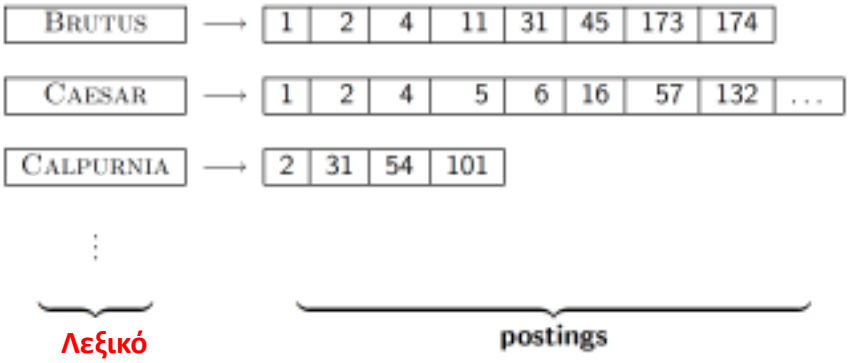
Διάλεξη 4: Κατασκευή Ευρετηρίου

Επανάληψη προηγούμενης διάλεξης

1. Δομές Δεδομένων για το Λεξικό
2. Ανάκτηση ανεκτική σε σφάλματα
   a. Ερωτήματα με *
   b. Διόρθωση ορθογραφικών λαθών
   c. Soundex

Δομές Δεδομένων για Λεξικά

Περιέχουν το *λεξιλόγιο όρων (λήμμα)*, τη συχνότητα εγγράφου (document frequency), δείκτες σε κάθε λίστα καταχωρήσεων, …
ποια δομή δεδομένων είναι κατάλληλη;



Δομές δεδομένων για το Λεξικό

- Δυο βασικές επιλογές:
  - Πίνακες Κατακερματισμού (Hashtables)
  - Δέντρα (Trees)
- Μερικά Συστήματα Ανάκτησης Πληροφορίας χρησιμοποιούν πίνακες κατακερματισμού άλλα δέντρα

Δομές Δεδομένων για Λεξικά

Κριτήρια Επιλογής

- Αποδοτική αναζήτηση ενός όρου (κλειδιού) στο λεξικό.
  - Σχετικές συχνότητας προσπέλασης των κλειδιών (πιο γρήγορα οι συχνοί όροι;)
- Πόσοι είναι οι όροι (κλειδιά)
- Είναι στατικό (ή έχουμε συχνά εισαγωγές/διαγραφές όρων) ή και τροποποιήσεις

Πίνακες Κατακερματισμού

Κάθε όρος του λεξιλογίου κατακερματίζεται σε έναν ακέραιο

+:

- Η αναζήτηση είναι πιο γρήγορη από ένα δέντρο: O(1)

- :

- Δεν υπάρχει εύκολος τρόπος να βρεθούν μικρές παραλλαγές ενός όρου
  - judgment/judgement, *resume* vs. *résumé*
- Μη δυνατή η προθεματική αναζήτηση [ανεκτική ανάκληση]
- Αν το λεξιλόγιο μεγαλώνει συνεχώς, ανάγκη για να γίνει κατακερματισμός από την αρχή

Δέντρα

- Το απλούστερο: δυαδικό δέντρο
- Το πιο συνηθισμένο: B-δέντρα
- Τα δέντρα απαιτούν να υπάρχει διάταξη των κλειδιών (αλλά συνήθως υπάρχει)

+:

- Λύνουν το πρόβλημα προθέματος (π.χ., όροι που αρχίζουν με *hyp*)

-:

- Πιο αργή: O(log *M*) [όπου M ο αριθμός των όρων- και αυτό απαιτεί (*ισοζυγισμένα balanced* δέντα]
- Η ισοζύγιση (rebalancing) των δυαδικών δέντρων είναι ακριβό
  - Αλλά τα B-δέντρα καλύτερα

7

Ερωτήματα με Wild-card (*)

Τρεις προσεγγίσεις:

1. B-δέντρο και αντεστραμμένο B-δέντρο
2. Permuterm index (ευρετήριο αντιμετατεθειμένων όρων)
3. *k*-gram index (ευρετήριο *k*-γραμμάτων)

8

Ερωτήματα με Wild-card (*)

- ***mon*:*** Βρες όλα τα έγγραφα που περιέχουν οποιαδήποτε λέξη *αρχίζει με "mon".*
  - Εύκολο όταν το λεξικό με δυαδικό δέντρο (ή B-δέντρο):
    1. Ανάκτησε όλους τους όρους t στο διάστημα: ***mon ≤ t < moo***
    2. Για κάθε όρο, αναζήτησε το αντεστραμμένο ευρετήριο σε ποια έγγραφα εμφανίζεται
- **** mon:*** Βρες όλα τα έγγραφα που περιέχουν οποιαδήποτε λέξη τελειώνει σε "mon": *πιο δύσκολο*
  - Διατήρησε ένα επιπρόσθετο B-tree για τους όρους ανάποδα *backwards (πχ ο όρος demon -> nomed)*
  - Ανάκτησε όλους τους όρους t στο διάστημα: ***nom ≤ t < non***.

Γενικά ερωτήματα με *

- * στη μέση του όρου
  - ***co*tion***
- Αναζήτησε το ***co**** AND ****tion*** σε ένα B-tree και υπολόγισε την τομή των συνόλων
  - Ακριβό!
- Εναλλακτική λύση: Μετάτρεψε τις ερωτήσεις έτσι ώστε τα * να εμφανίζονται στο τέλος

Permuterm Index (ευρετήριο αντιμετατεθειμένων όρων)

Ευρετήριο Permuterm

Βασική ιδέα: Δεξιά περιστροφή (rotation) του όρου του ερωτήματος ώστε το * στο τέλος

π.χ., Ερώτημα he*lo -> he*lo$ -> lo$he*

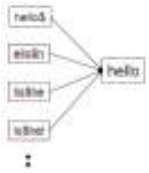
όπου $ ένα ειδικός χαρακτήρας που σηματοδοτεί το τέλος μιας λέξης

Ψάχνουμε το lo$hel*

Κατασκευάζουμε ένα ευρετήριο αντιμετατεθειμένων όρων στο οποίο οι διάφορες παραλλαγές που προκύπτουν από την περιστροφή του όρου συνδέονται με τον αρχικό όρο

Πχ. για τον όρο **hello -> hello$**, εισάγουμε στο ευρετήριο τα:

- **hello$, o$hell , lo$hel (match), llo$he, ello$h**



Ευρετήριο Permuterm

- **X*Y*Z** πως γίνεται match?
  - X*Y*Z$ -> Z$X*
  - Ψάξε Z$X* και μετά έλεγξε κάθε υποψήφιο όρο για το Y
  - Πχ fi*mo*er -> ψάξε er$fi*, έλεγξε αν και mo (π.χ., fishmonger και fillbuster)
- Στην πραγματικότητα, permuterm B-tree
- *Πρόβλημα: ≈ τετραπλασιάζει το μέγεθος του λεξικού*

Εμπειρική παρατήρηση για τα Αγγλικά

Ευρετήρια *k*-γραμμάτων (*k*-gram indexes)

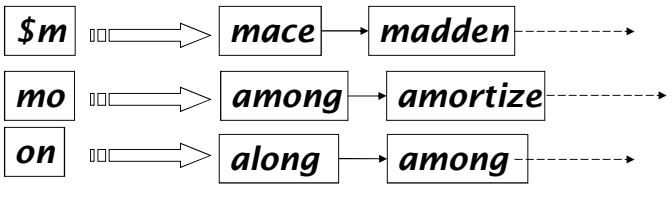
- Απαρίθμησε όλα τα *k*-γράμματα (ακολουθίες *k* γραμμάτων) που εμφανίζονται σε κάθε όρο
  - *π.χ.,* για το κείμενο "***April is the cruelest month***" έχουμε τα 2-γράμματα (*bigrams*)

    a,ap,pr,ri,il,l,i,is,s,t,th,he,e,$c,cr,ru, ue,el,le,es,st,t$, m,mo,on,nt,h

    - Όπου $ ένα ειδικός χαρακτήρας που σηματοδοτεί το τέλος και την αρχή μιας λέξης
- Διατήρησε ένα <u>δεύτερο</u> αντεστραμμένο ευρετήριο από τα 2-γράμματα στους όρους του λεξικού που τα περιέχουν

Ευρετήρια *k*-γραμμάτων (*k*-gram indexes)

- Το ευρετήριο *k*-γραμμάτων βρίσκει τους όρους βασισμένο σε μια ερώτηση που αποτελείται από *k*-γράμματα (εδώ *k*=2).



k = 3

Ευρετήρια *k*-γραμμάτων (*k*-gram indexes)

- Ερώτημα ***mon**** τώρα γίνεται

  $m *AND* ***mo*** *AND* ***on***

  - Βρίσκει τους όρους που ταιριάζουν μια AND εκδοχή του ερωτήματος
- Απαιτείται βήμα μετά-φιλτραρίσματος (post-filter)
  - False positive, π.χ., moon
- Οι όροι που απομένουν αναζητούνται στο γνωστό αντεστραμμένο ευρετήριο όρων-εγγράφων

Επεξεργασία ερωτημάτων

- Π.χ., Θεωρείστε το ερώτημα:

  se*ate *AND* ***fil*er***

Μπορεί να οδηγήσει στην εκτέλεση πολλών Boolean *AND* ερωτημάτων (πιθανοί συνδυασμοί όρων).

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 τις λέξεις γύρω
      - π.χ., ***I flew form Heathrow to Narita.***

Διόρθωση εγγράφων

- Χρήσιμη ιδιαίτερα για έγγραφα μετά από OCR
  - Αλγόριθμοι διόρθωσης ρυθμισμένοι για αυτό: rn/m
  - Μπορεί να χρησιμοποιούν ειδική γνώση (domain-specific)
    - Π.χ., OCR μπερδεύει το O με το D πιο συχνά από το O και το I (που είναι γειτονικά στα QWERTY πληκτρολόγιο, οπότε πιο πιθανή η ανταλλαγή τους στην πληκτρολόγηση)
- Αλλά συχνά: web σελίδες αλλά και τυπωμένο υλικό έχουν typos
- Στόχος: το λεξικό να περιέχει λιγότερα ορθογραφικά λάθη
- Αλλά συχνά δεν αλλάζουμε τα έγγραφα αλλά επεκτείνουμε την απεικόνιση ερωτήματος –εγγράφου

Διόρθωση λαθών στο ερώτημα

- Μπορεί είτε
  - Να ανακτήσουμε τα έγγραφα που έχουν δεικτοδοτηθεί κάτω από τη σωστή ορθογραφία, Ή
  - Να επιστρέψουμε διάφορες προτεινόμενα ερωτήματα με σωστή ορθογραφία
    - *Did you mean … ?*

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)

Διόρθωση μεμονωμένης λέξης

Δοθέντος ενός ***Λεξικού*** και ένα ερωτήματος Q, επέστρεψε τις λέξεις του λεξικού που είναι πιο κοντά στο Q

- Τι σημαίνει "πιο κοντά"?
- Διαφορετικοί ορισμοί εγγύτητας:
  - Την απόσταση διόρθωσης -- edit distance (Levenshtein distance) και την σταθμισμένη απόσταση διόρθωσης -- weighted edit distance
  - 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

- Συνήθως, οι πράξεις είναι σε επίπεδο χαρακτήρα
  - **Levenshtein distance:** (1) Insert – Εισαγωγή, (2) Delete - Διαγραφή και (3) Replace – Αντικατάσταση ενός χαρακτήρα
  - **Damerau-Levenshtein distance:** + Transposition - Αντιμετάθεση ένα χαρακτήρα
- Π.χ., η απόσταση διόρθωσης από ***dof*** σε ***dog*** είναι 1
  - Από ***cat*** σε ***act*** είναι 2 (Μόνο 1 με αντιμετάθεση)
  - Από ***cat*** σε ***dog*** είναι 3.

Δυναμικός προγραμματισμός

- ✓ Εκφράζουμε το πρόβλημα ως συνδυασμό υπό-προβλημάτων – η βέλτιστη λύση βασίζεται στη βέλτιστη λύση υπό-πρόβληματος
  - ✓ Στην περίπτωση των αποστάσεων διόρθωσης – το υπό-πρόβλημα δυο προθεμάτων:

    Ο βέλτιστος τρόπος από μια λέξη σε μια άλλη, βασίζεται στο βέλτιστο τρόπο από κάποιο πρόθεμα της πρώτης σε πρόθεμα της δεύτερης
- ✓ Οι επικαλυπτόμενες υπό-λύσεις: χρειαζόμαστε τις περισσότερες αποστάσεις 3 φορές: κίνηση δεξιά, στη διαγώνιο, κάτω

Υπολογισμός απόστασης Levenshtein

	j -1	j
i -1	cost from upper left neighbor (optimal m[i-1,j-1]) To get m[i, j] either copy if s1[i]=s2[j] +0, or else replace +1 [i-1, j-1]	Cost from upper neighbor (delete) [i-1, j]
i	cost from left neighbor (insert) [i, j-1]	the minimum of the three possible "movements"; the cheapest way of getting here [i, j]

Levenshtein distance: Algorithm

```
LevenshteinDistance(s1, s2)
 1  for i ← 0 to |s1|
 2  do m[i, 0] = i
 3  for j ← 0 to |s2|
 4  do m[0, j] = j
 5  for i ← 1 to |s1|
 6  do for j ← 1 to |s2|
 7      do if s1[i] = s2[j]
 8            then m[i, j] = min{m[i-1, j]+1, m[i, j-1]+1, m[i-1, j-1]}
 9            else  m[i, j] = min{m[i-1, j]+1, m[i, j-1]+1, m[i-1, j-1]+1}
10  return m[|s1|, |s2|]
```

Operations: insert (cost 1), delete (cost 1), replace (cost 1), copy (cost 0)

Levenshtein distance: Algorithm

```
LevenshteinDistance(s1, s2)
 1  for i ← 0 to |s1|
 2  do m[i, 0] = i
 3  for j ← 0 to |s2|
 4  do m[0, j] = j
 5  for i ← 1 to |s1|
 6  do for j ← 1 to |s2|
 7      do if s1[i] = s2[j]
 8            then m[i, j] = min{m[i-1, j]+1, m[i, j-1]+1, m[i-1, j-1]}
 9            else  m[i, j] = min{m[i-1, j]+1, m[i, j-1]+1, m[i-1, j-1]+1}
10  return m[|s1|, |s2|]
```

Operations: insert (cost 1), delete (cost 1), replace (cost 1), copy (cost 0)

Levenshtein distance: Algorithm

```
LevenshteinDistance(s1, s2)
 1  for i ← 0 to |s1|
 2  do m[i, 0] = i
 3  for j ← 0 to |s2|
 4  do m[0, j] = j
 5  for i ← 1 to |s1|
 6  do for j ← 1 to |s2|
 7       do if s1[i] = s2[j]
 8            then m[i, j] = min{m[i-1, j]+1, m[i, j-1]+1, m[i-1, j-1]}
 9            else m[i, j] = min{m[i-1, j]+1, m[i, j-1]+1, m[i-1, j-1]+1}
10  return m[|s1|, |s2|]
```

Operations: insert (cost 1), delete (cost 1), replace (cost 1), copy (cost 0)

Levenshtein distance: Algorithm

```
LevenshteinDistance(s1, s2)
 1  for i ← 0 to |s1|
 2  do m[i, 0] = i
 3  for j ← 0 to |s2|
 4  do m[0, j] = j
 5  for i ← 1 to |s1|
 6  do for j ← 1 to |s2|
 7       do if s1[i] = s2[j]
 8            then m[i, j] = min{m[i-1, j]+1, m[i, j-1]+1, m[i-1, j-1]}
 9            else m[i, j] = min{m[i-1, j]+1, m[i, j-1]+1, m[i-1, j-1]+1}
10  return m[|s1|, |s2|]
```

Operations: insert (cost 1), delete (cost 1), replace (cost 1), copy (cost 0)

Υπολογισμός απόστασης: παράδειγμα

cat -> cart

Σταθμισμένη απόσταση διόρθωσης

- Το βάρος μιας πράξης εξαρτάται από τον ποιο χαρακτήρα (χαρακτήρες) περιλαμβάνει
  - Στόχος να λάβει υπόψη λάθη OCR ή πληκτρολόγησης Παράδειγμα: ***m*** πιο πιθανό να πληκτρολογηθεί ως ***n*** παρά ως ***q***
  - Οπότε η αντικατάσταση του ***m*** από ***n*** έχει μικρότερη απόσταση διόρθωσης από την απόσταση του από το ***q***
- Προϋποθέτει ως είσοδος ένας πίνακας βαρών
- *Πως θα μετατρέψουμε το δυναμικό προγραμματισμό για να χειριστούμε τα βάρη;*

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

- Απαρίθμησε όλα το *k*-γράμματα στον όρο της ερώτησης
- Χρησιμοποίησε το ευρετήριο *k*-γραμμάτων για να ανακτήσεις όλους τους όρους του λεξικού που ταιριάζουν *κάποιο* (>= *κατώφλι*) αριθμό από τα *k*-γράμματα του ερωτήματος

Παράδειγμα με 3-γράμματα

- Έστω ότι ο όρος στο λεξικό είναι ***november***
  - Τα τριγράμματα είναι *nov, ove, vem, emb, mbe, ber*.
- Για το ερώτημα ***december***
  - Τα τριγράμματα είναι *dec, ece, cem, emb, mbe, ber*.
- Άρα επικαλύπτονται 3 τριγράμματα (από τα 6 κάθε όρου)

Επικάλυψη *k*-γραμμάτων

- Συνήθης μέτρηση της επικάλυψης

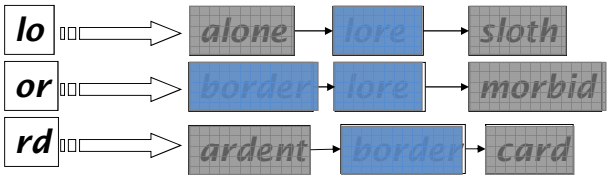
Έστω *X* και *Y* δύο σύνολα, ο συντελεστής Jaccard (J.C.) ορίζεται ως:

$$|X \cap Y| / |X \cup Y|$$

  - Ίσος με 1 όταν τα *X* και *Y* έχουν τα ίδια στοιχεία και 0 όταν είναι ξένα
- Τα *X* and *Y* δε χρειάζεται να έχουν το ίδιο μέγεθος
- Πάντα μεταξύ του 0 και του 1
  - Το κατώφλι καθορίζει αν υπάρχει ταίριασμα, πχ., αν J.C. > 0.8, τότε ταίριασμα

Επικάλυψη *k*-γραμμάτων

- Έστω το ερώτημα ***lord*** – θέλουμε να βρούμε τις λέξεις που ταιριάζουν 2 από τα 3 2-γράμματα (***lo, or, rd***)



Διόρθωση βασισμένη στα συμφραζόμενα

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 υπήρχαν έγγραφα που να ταιριάζουν το 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

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

ΦΩΝΗΤΙΚΗ ΔΙΟΡΘΩΣΗ (SOUNDEX)

Soundex

- Κλάση ευριστικών για την επέκταση ενός ερωτήματος σε φωνητικά (phonetic) ισοδύναμα
  - Εξαρτώνται από τη γλώσσα – κυρίως για ονόματα
    - Π.χ., ***chebyshev*** → ***tchebycheff***
- Προτάθηκε από το U.S. census … το 1918

Soundex – τυπικός αλγόριθμος

- Μετάτρεψε κάθε token προς δεικτοδότηση σε μια μορφή 4-χαρακτήρων
- Το ίδιο και για τους όρους του ερωτήματος
- Κατασκεύασε και ψάξε στο ευρετήριο τις μειωμένες μορφές
  - (όταν το ερώτημα χρειάζεται φωνητικό ταίριασμα)
- http://www.creativyst.com/Doc/Articles/SoundEx1/SoundEx1.htm#Top

Soundex – τυπικός αλγόριθμος

1. Κράτησε τον πρώτο χαρακτήρα της λέξης
2. Μετάτρεψε όλες τις εμφανίσεις των παρακάτω όρων σε '0' (zero):
   'A', E', 'I', 'O', 'U', 'H', 'W', 'Y'.
3. Άλλαξε τα γράμματα σε αριθμούς ως ακολούθως:
   - B, F, P, V $\rightarrow$ 1
   - C, G, J, K, Q, S, X, Z $\rightarrow$ 2
   - D,T $\rightarrow$ 3
   - L $\rightarrow$ 4
   - M, N $\rightarrow$ 5
   - R $\rightarrow$ 6

Soundex continued

4. Σβήσε όλα τα ζεύγη συνεχόμενων αριθμών
5. Σβήσε όλα τα υπομένοντα 0
6. Πρόσθεσε 0 στο τέλος και επέστρεψε τις τέσσερις πρώτες θέσεις που θα είναι της μορφής <uppercase letter> <digit> <digit> <digit>.

Π.χ., ***Herman*** γίνεται H655.

Το ***hermann*** δίνει τον ίδιο κωδικό;

ΤΕΛΟΣ ΕΠΑΝΑΛΗΨΗΣ

Τι θα δούμε σήμερα;

- Κατασκευή του Ευρετηρίου

Κατασκευή ευρετηρίου

- Πως κατασκευάζουμε το ευρετήριο;
- Ποιες στρατηγικές χρησιμοποιούμε όταν έχουμε περιορισμένη κυρίως 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).
- 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

REUTERS

You are here: Home > News > Science > Article

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

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.

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

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. . .

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 θέση - 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;

- Διαπέραση 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

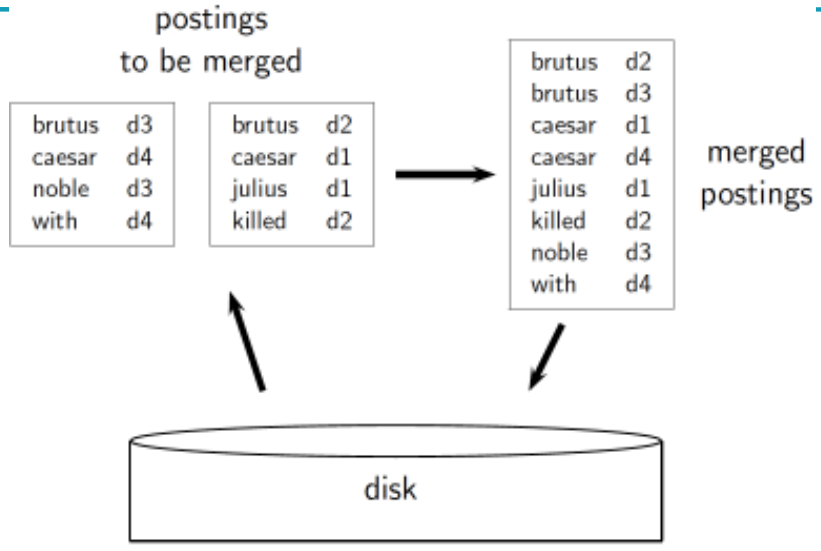
Αν κάθε σύγκριση χρειάζεται 2 προσπελάσεις στο δίσκο, και για τη διάταξη N στοιχείων χρειαζόμαστε N $\log_2 N$ συγκρίσεις, πόσο χρόνο θα χρειαζόμασταν;

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

- 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.
  - Μετά συγχώνευσε τα blocks σε ένα μεγάλο διατεταγμένο block.

59

Παράδειγμα



60

Διάταξη 10 blocks των 10M εγγραφών

- Πρώτα, διάβασε κάθε block και διάταξε τις εγγραφές του:
  - Quicksort 2*N* ln *N* expected steps
  - 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

61

Διάταξη 10 blocks των 10M εγγραφών

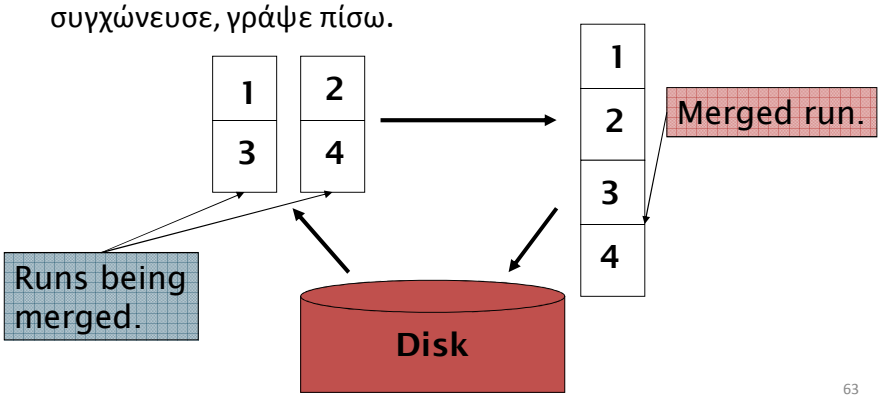
```
BSBINDEXCONSTRUCTION()
1  n ← 0
2  while (all documents have not been processed)
3  do n ← n + 1
4     block ← PARSENEXTBLOCK()
5     BSBI-INVERT(block)
6     WRITEBLOCKTODISK(block, f_n)
7  MERGEBLOCKS(f_1, ..., f_n; f_merged)
```

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

62

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

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)

- *Βασική Ιδέα 1:* Δημιουργία ξεχωριστών λεξικών για κάθε block – δε χρειάζεται να διατηρούμε term-termID απεικονίσεις μεταξύ blocks.
- *Βασική Ιδέα 2*: *Αποφυγή της διάταξης των όρων.* Συγκεντρώσετε τις καταχωρήσεις σε λίστες καταχωρήσεων όπως αυτές εμφανίζονται.
- *Κατασκευή ενός πλήρους αντεστραμμένου ευρετηρίου για κάθε block.*
- 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 (οπότε οι καταχωρήσεις για τον ίδιο όρο στον ίδιο «κάδο»

- Η συγχώνευση όπως και στο BSBI.

67

Δυναμικά ευρετήρια

- Μέχρι στιγμής, θεωρήσαμε ότι τα ευρετήρια είναι στατικά.
- Αυτό συμβαίνει σπάνια, στην πραγματικότητα:
  - Νέα έγγραφα εμφανίζονται και πρέπει να εισαχθούν
  - 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

68

Η πιο απλή προσέγγιση

- Διατήρησε ένα «μεγάλο» κεντρικό ευρετήριο
- Τα νέα έγγραφα σε μικρό «βοηθητικό» ευρετήριο (στη μνήμη)
- Ψάξε και στα δύο, συγχώνευσε το αποτέλεσμα
- Διαγραφές
  - Invalidation bit-vector για τα διαγραμμένα έγγραφα
- 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 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
- Κρατάμε το μικρότερο (Z_0) στη μνήμη
- Τα μεγαλύτερα (I_0, I_1, …) στο δίσκο

- Όταν το Z_0 γίνει πολύ μεγάλο (> n),
- το γράφουμε στο δίσκο ως I_0 ή
- Ή το συγχωνεύουμε με το I_0 ως Z_1 (αν το I_0 υπάρχει ήδη)
- Ή γράφουμε στο δίσκο το Z_1 to disk ως I_1 (αν δεν υπάρχει το I_1)
- Ή συγχώνευση με το I_1 ως Z_2

71


```
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())
```

72

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

Κατανεμημένη κατασκευή

- Για ευρετήριο κλίμακας web (don't try this at home!):
  Χρήση κατανεμημένου cluster
- Επειδή μια μηχανή είναι επιρρεπής σε αποτυχία
  - Μπορεί απροσδόκητα να γίνει αργή ή να αποτύχει
- Χρησιμοποίηση πολλών μηχανών

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 στο επόμενο μάθημα

ΤΕΛΟΣ 4ου Μαθήματος

Ερωτήσεις?

Χρησιμοποιήθηκε κάποιο υλικό των:
✓Pandu Nayak and Prabhakar Raghavan, CS276:Information Retrieval and Web Search (Stanford)
✓Hinrich Schütze and Christina Lioma, Stuttgart IIR class