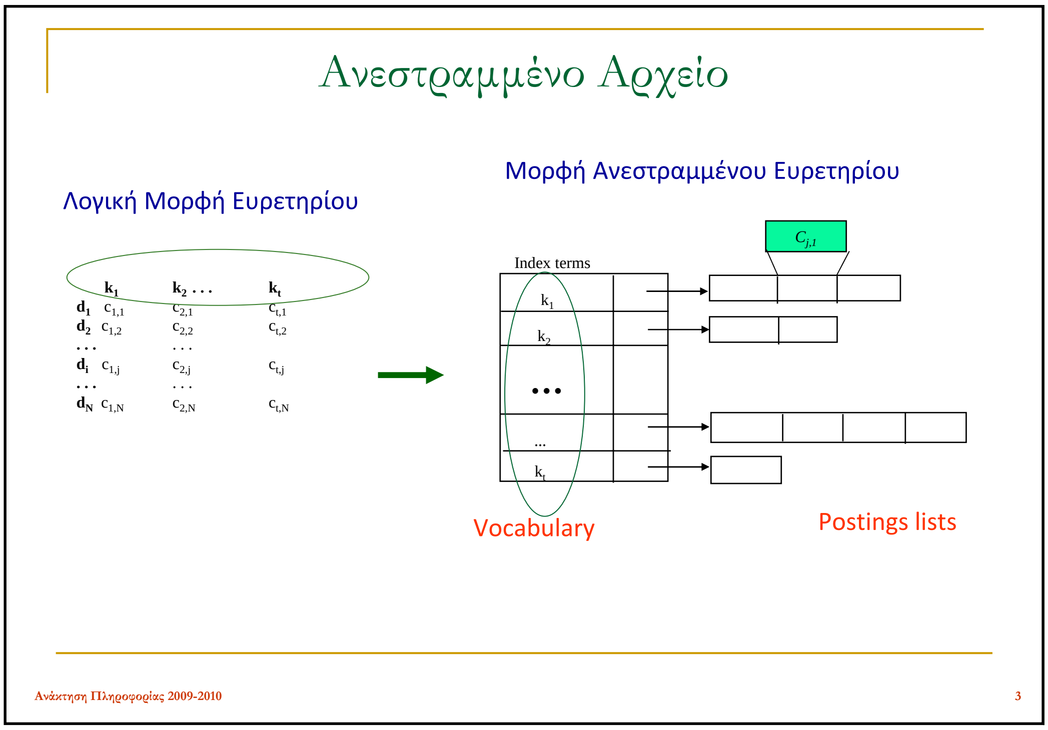
Ευρετηρίαση
ΜΕΡΟΣ ΙΙΙ

Επεξεργασία Κειμένου

Content

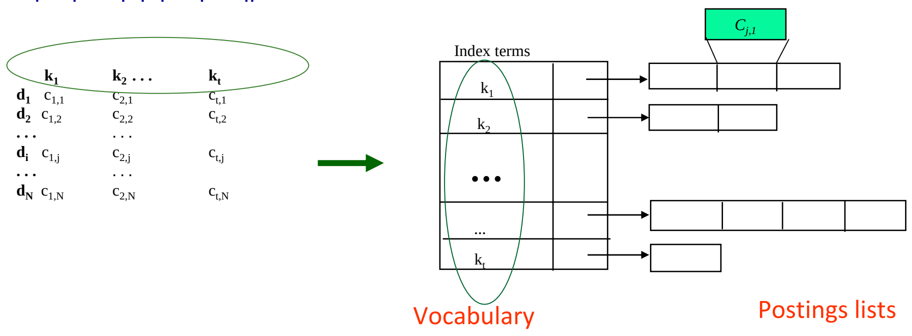
- Recap: Faster posting lists with skip pointers, Phrase and Proximity Queries, Dictionary
- Wild-Card Queries
  - Permutex
  - *k*-gram indexes
- Spelling Corrections
- Pattern Matching



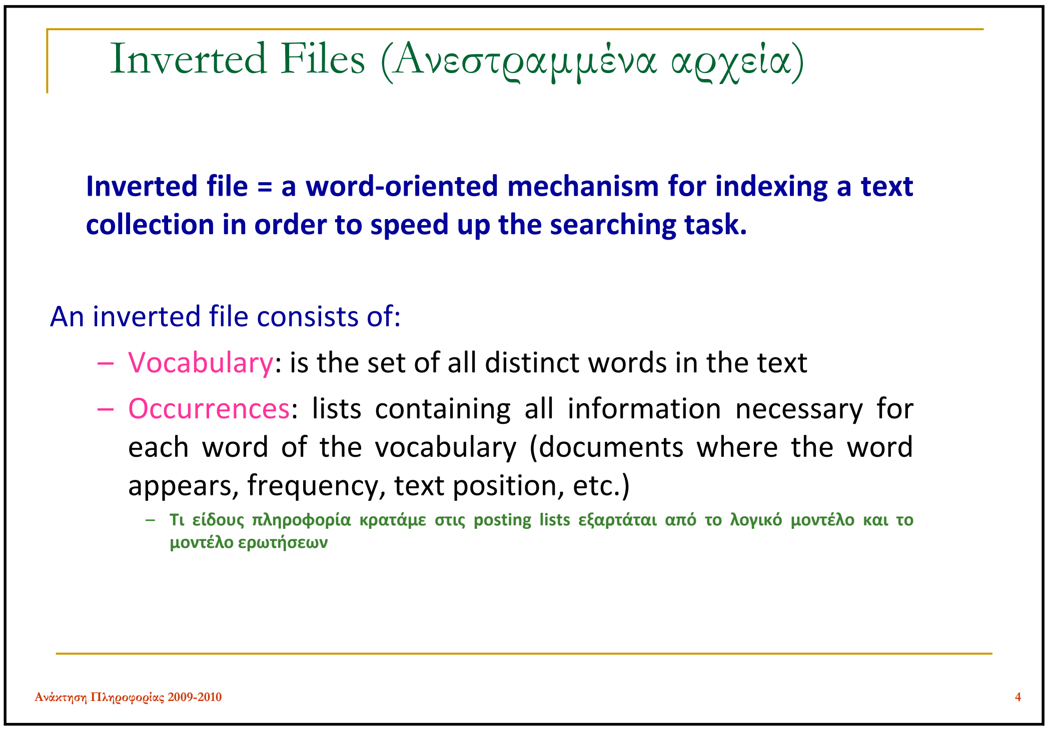
Ανεστραμμένο Αρχείο

Λογική Μορφή Ευρετηρίου

Μορφή Ανεστραμμένου Ευρετηρίου



k_1 k_2 ... k_t
d_1 $c_{1,1}$ $c_{2,1}$ $c_{t,1}$
d_2 $c_{1,2}$ $c_{2,2}$ $c_{t,2}$
...
d_i $c_{1,j}$ $c_{2,j}$ $c_{t,j}$
...
d_N $c_{1,N}$ $c_{2,N}$ $c_{t,N}$
$C_{j,1}$
Index terms
k_1
k_2
•••
...
k_t
Vocabulary
Postings lists



Inverted Files (Ανεστραμμένα αρχεία)

Inverted file = a word-oriented mechanism for indexing a text collection in order to speed up the searching task.

An inverted file consists of:

- Vocabulary: is the set of all distinct words in the text
- Occurrences: lists containing all information necessary for each word of the vocabulary (documents where the word appears, frequency, text position, etc.)
  - **Τι είδους πληροφορία κρατάμε στις posting lists εξαρτάται από το λογικό μοντέλο και το μοντέλο ερωτήσεων**

Searching an inverted index

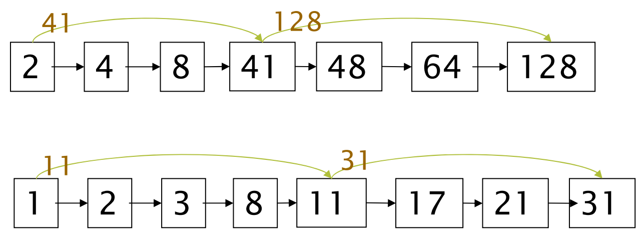
General Steps:

1. Vocabulary search:
   the words present in the query are searched in the vocabulary

2. Retrieval occurrences:
   the lists of the occurrences of all words found are retrieved

3. Manipulation of occurrences:
   The occurrences are processed to solve the query

Augment postings with skip pointers (at indexing time)



Why?

- To skip postings that will not figure in the search results.

1. How?
2. Where do we place skip pointers?

Phrase queries

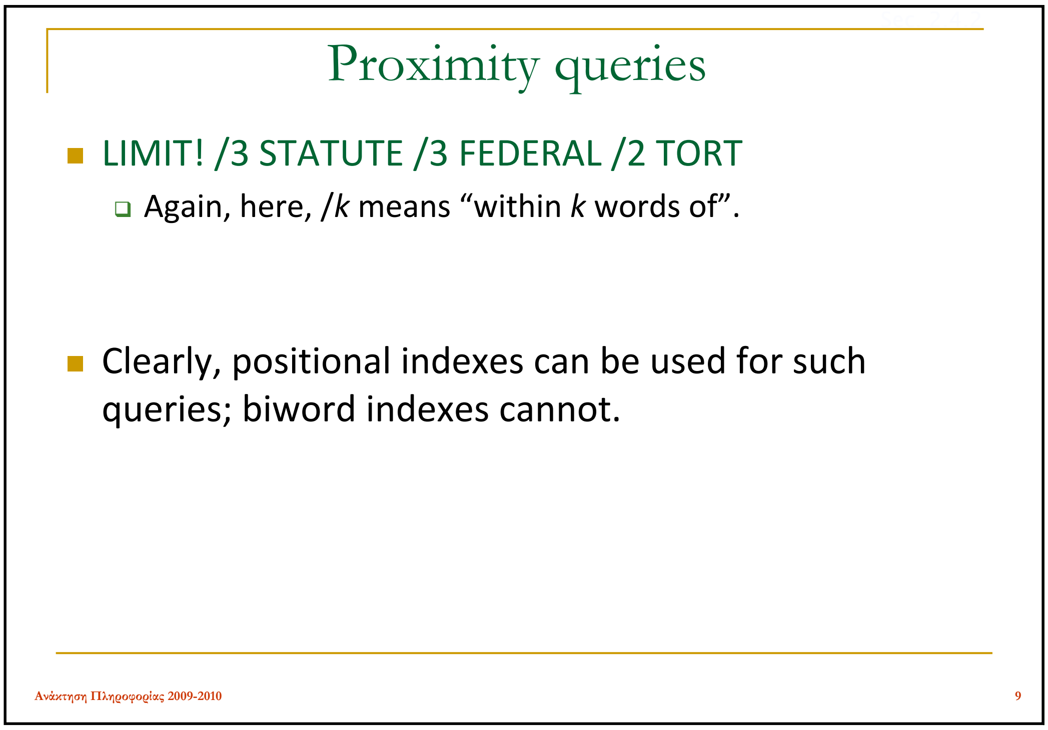
- Want to be able to answer queries such as “***stanford university”*** – as a phrase
- Why?
  - The concept of phrase queries has proven easily understood by users; one of the few “advanced search” ideas that works -- 10% explicit phrase queries (“”)
  - Many more queries are *implicit phrase queries* (such as person names)

1. Biword Indexes:

Index every consecutive pair of terms in the text as a phrase

2. Positional Indexes:

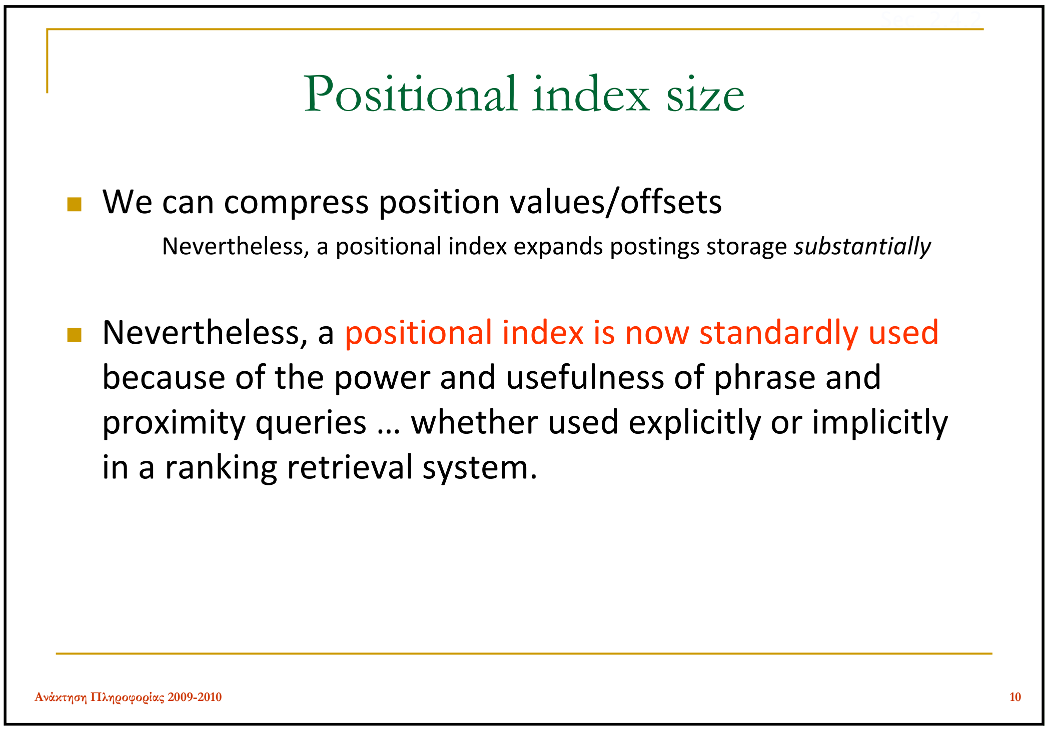
In the postings, store, for each ***term*** the position(s) in which tokens of it appear



Proximity queries

- LIMIT! /3 STATUTE /3 FEDERAL /2 TORT
  - Again, here, /*k* means “within *k* words of”.
- Clearly, positional indexes can be used for such queries; biword indexes cannot.

Ανάκτηση Πληροφορίας 2009-2010 9

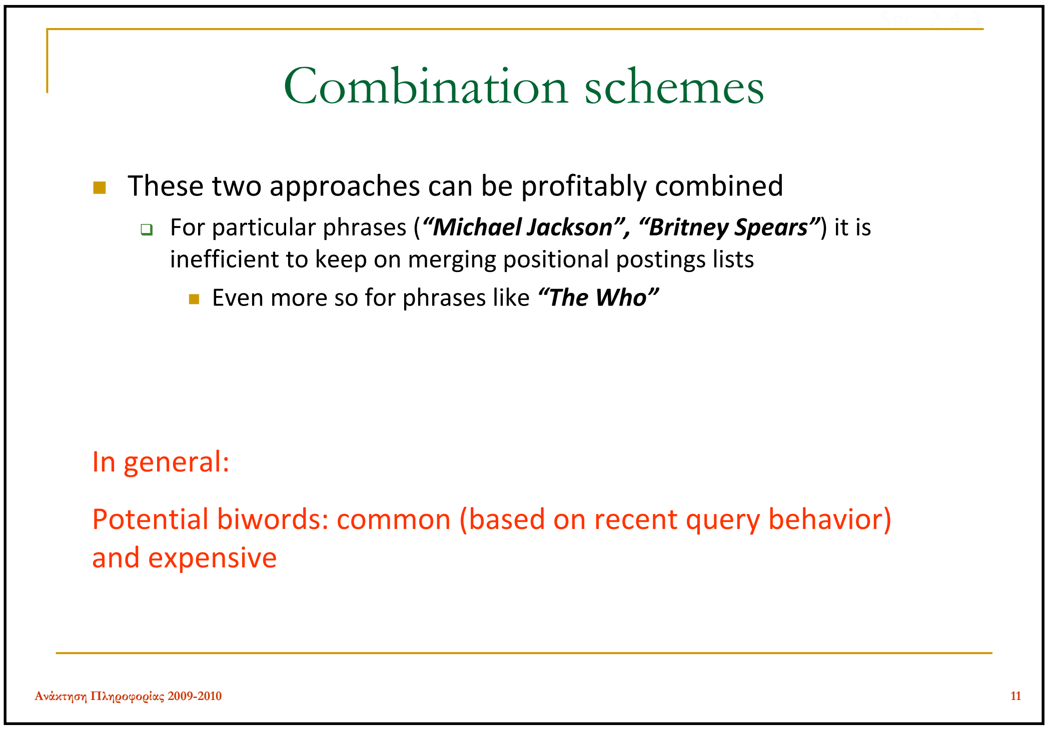


Positional index size

- We can compress position values/offsets

  Nevertheless, a positional index expands postings storage *substantially*
- Nevertheless, a positional index is now standardly used because of the power and usefulness of phrase and proximity queries … whether used explicitly or implicitly in a ranking retrieval system.

Ανάκτηση Πληροφορίας 2009-2010 10



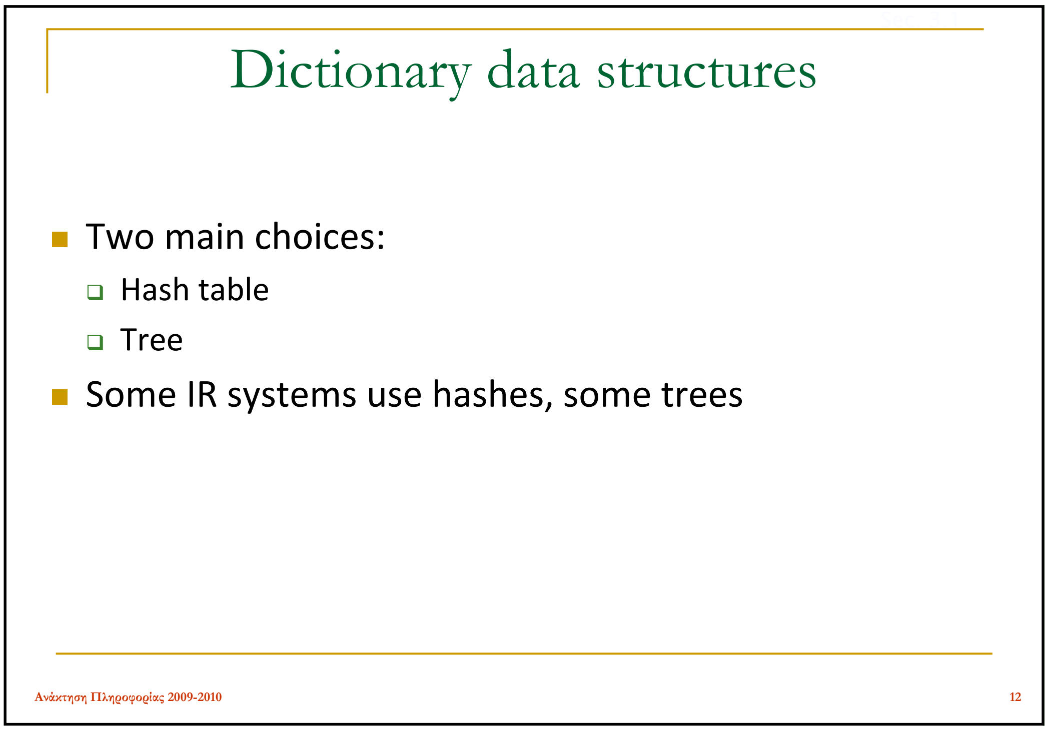
Combination schemes

- These two approaches can be profitably combined
  - For particular phrases (***"Michael Jackson", "Britney Spears"***) it is inefficient to keep on merging positional postings lists
    - Even more so for phrases like ***"The Who"***

In general:

Potential biwords: common (based on recent query behavior) and expensive

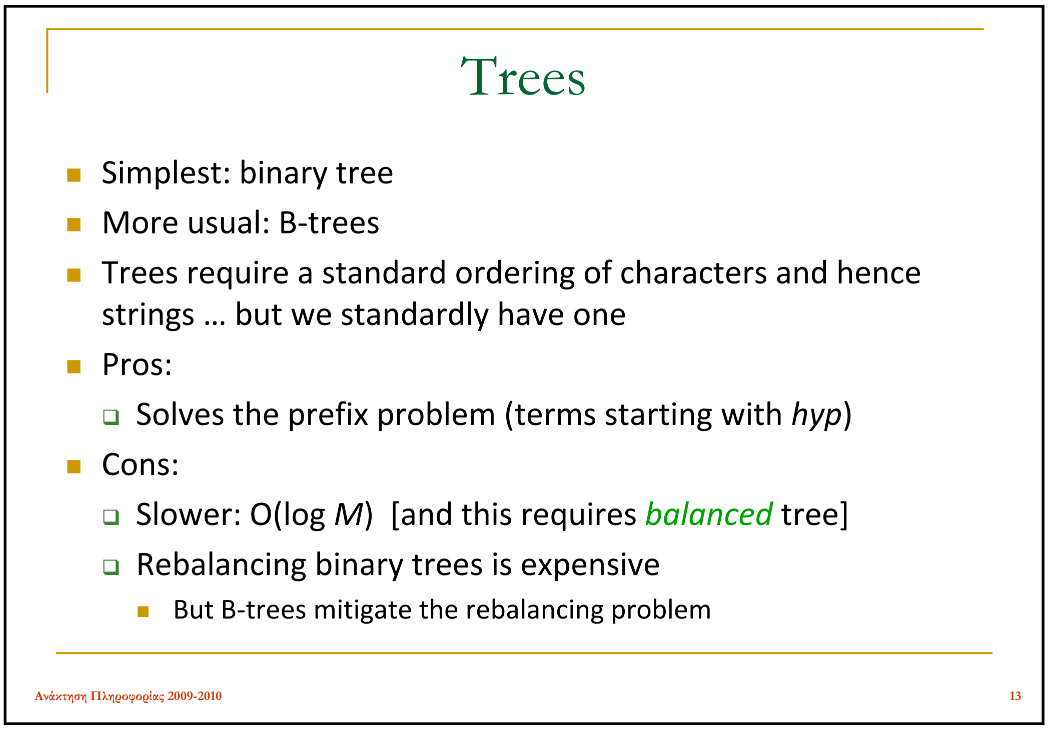
Ανάκτηση Πληροφορίας 2009-2010 11



Dictionary data structures

- Two main choices:
  - Hash table
  - Tree
- Some IR systems use hashes, some trees

Ανάκτηση Πληροφορίας 2009-2010 12

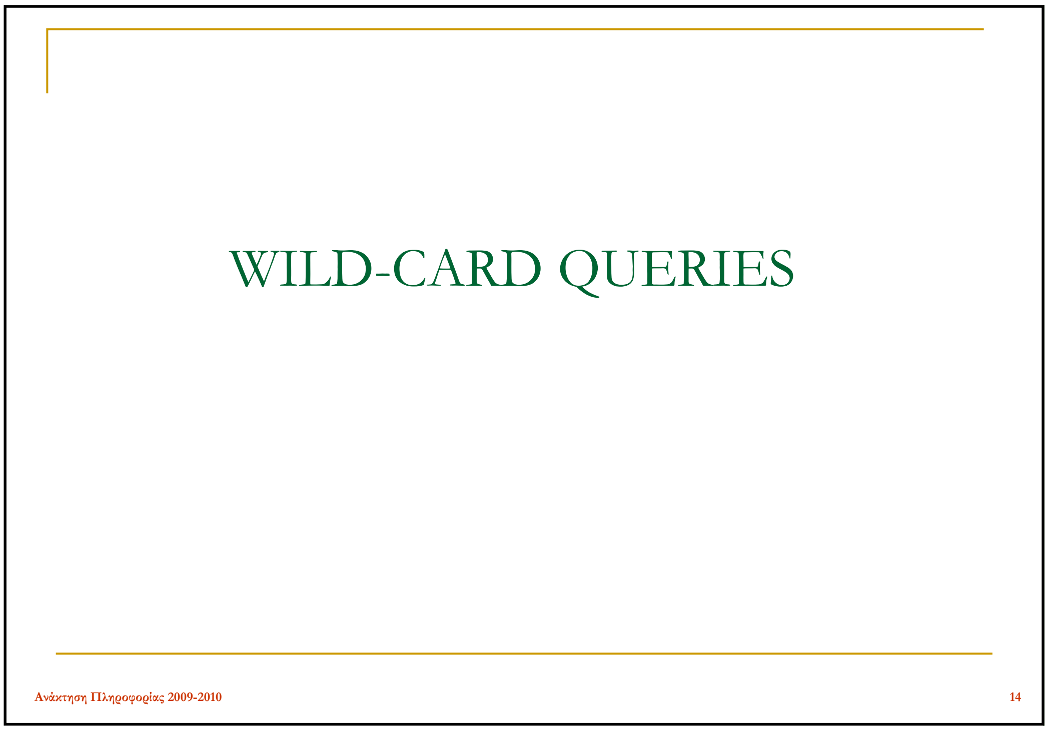


Sec. 3.1

Trees

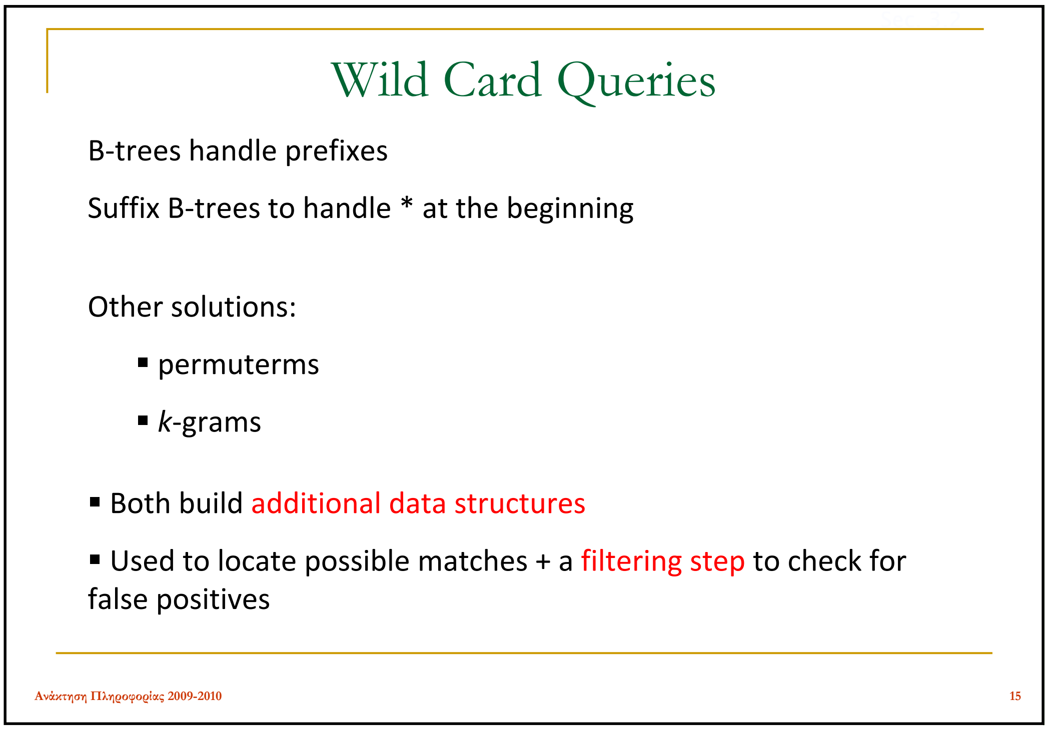
- Simplest: binary tree
- More usual: B-trees
- Trees require a standard ordering of characters and hence strings … but we standardly have one
- Pros:
  - Solves the prefix problem (terms starting with *hyp*)
- Cons:
  - Slower: O(log *M*) [and this requires *balanced* tree]
  - Rebalancing binary trees is expensive
    - But B-trees mitigate the rebalancing problem

Ανάκτηση Πληροφορίας 2009-2010 13



WILD-CARD QUERIES

Ανάκτηση Πληροφορίας 2009-2010 14



Wild Card Queries

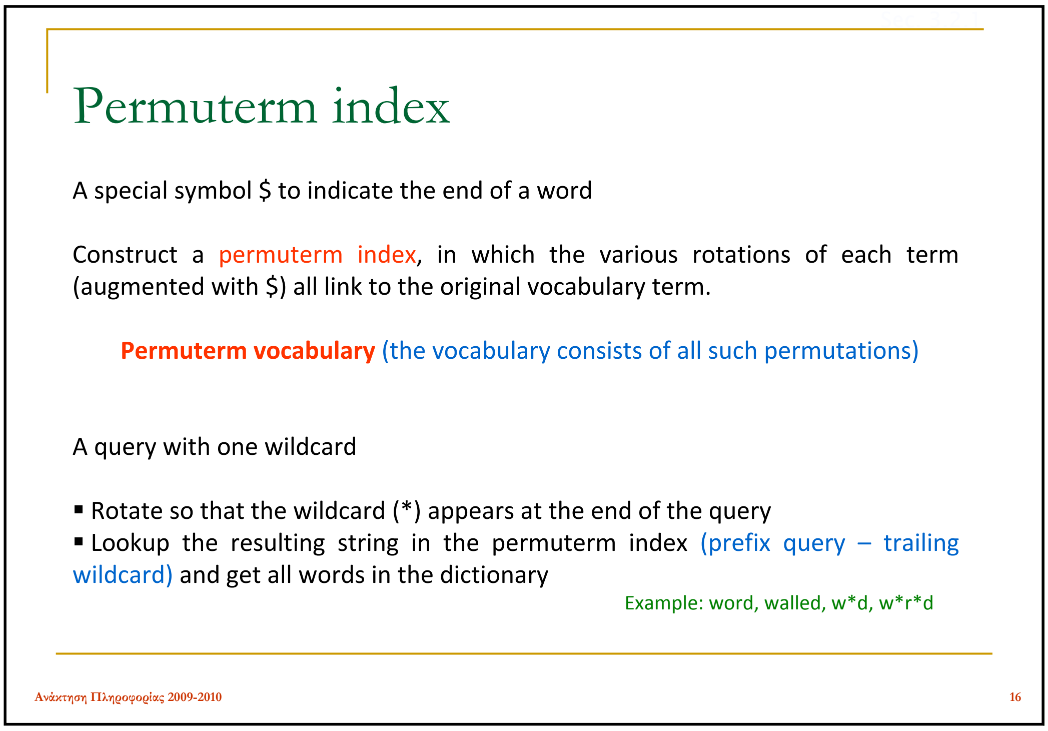
B-trees handle prefixes

Suffix B-trees to handle * at the beginning

Other solutions:

- permuterms
- *k*-grams

- Both build additional data structures
- Used to locate possible matches + a filtering step to check for false positives



Permuterm index

A special symbol $ to indicate the end of a word

Construct a permuterm index, in which the various rotations of each term (augmented with $) all link to the original vocabulary term.

Permuterm vocabulary (the vocabulary consists of all such permutations)

A query with one wildcard

- Rotate so that the wildcard (*) appears at the end of the query
- Lookup the resulting string in the permuterm index (prefix query – trailing wildcard) and get all words in the dictionary

Example: word, walled, w*d, w*r*d

Bigram (*k*-gram) index

A *k-gram* is a sequence of *k* characters

- Use as special character $ to denote the beginning or the end of a term
- In a *k*-gram index, the dictionary contains all *k*-grams that occur in any term in the vocabulary

- Maintain a *second inverted index from bigrams to dictionary terms* that match each bigram.
- Gets terms that match AND version of our wildcard query.

Example: word, walled, w*d, w*r*d

SPELLING CORRECTION

Spell correction

- Britney Spears

Britian spears
Britney's spears
Brandy spears
Pritanny spears (brittany spears)

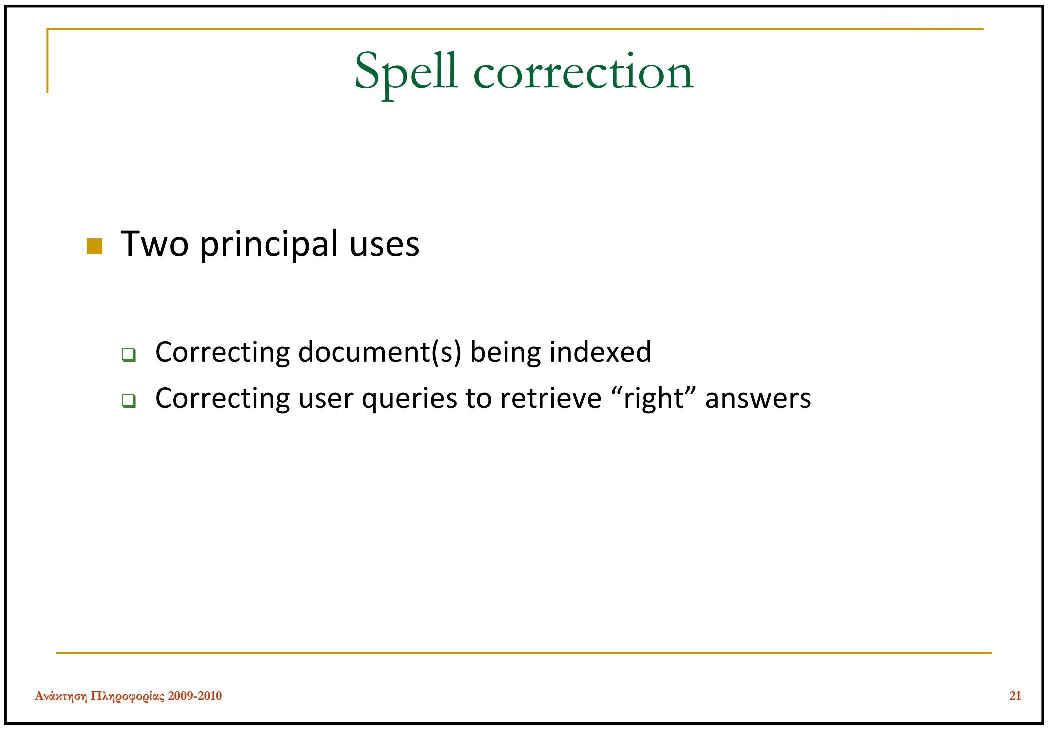
(all corrected in Google)

Spell correction

Two principles for correcting:

1. Of various alternatives: choose the nearest one (proximity, distance)

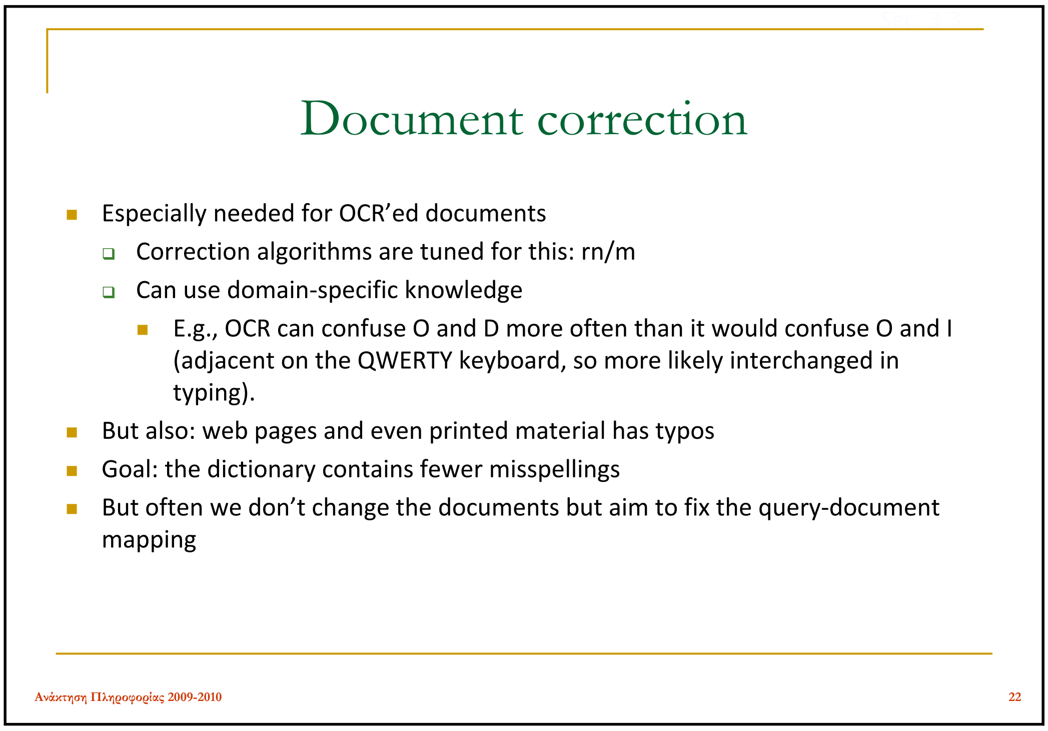
2. When two correctly spelled queries are (nearly) tied, select the most common one (common? #occurrences in the collection, most common among queries typed in by the users)



Spell correction

- Two principal uses
  - Correcting document(s) being indexed
  - Correcting user queries to retrieve “right” answers

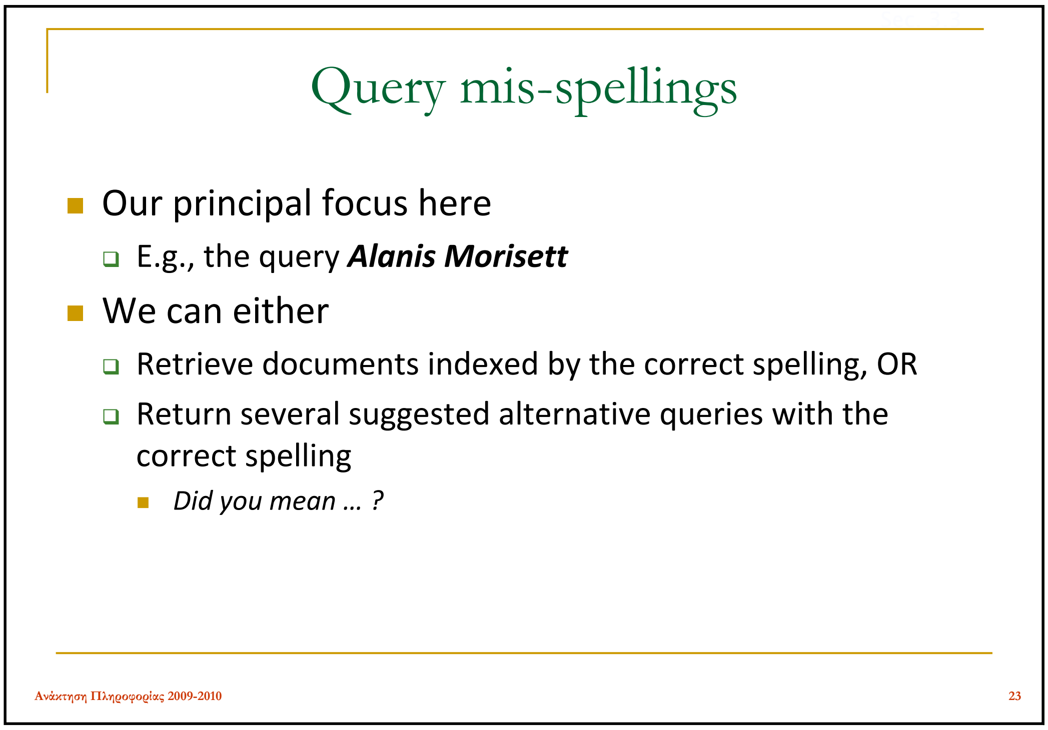
Ανάκτηση Πληροφορίας 2009-2010 21



Document correction

- Especially needed for OCR’ed documents
  - Correction algorithms are tuned for this: rn/m
  - Can use domain-specific knowledge
    - E.g., OCR can confuse O and D more often than it would confuse O and I (adjacent on the QWERTY keyboard, so more likely interchanged in typing).
- But also: web pages and even printed material has typos
- Goal: the dictionary contains fewer misspellings
- But often we don’t change the documents but aim to fix the query-document mapping

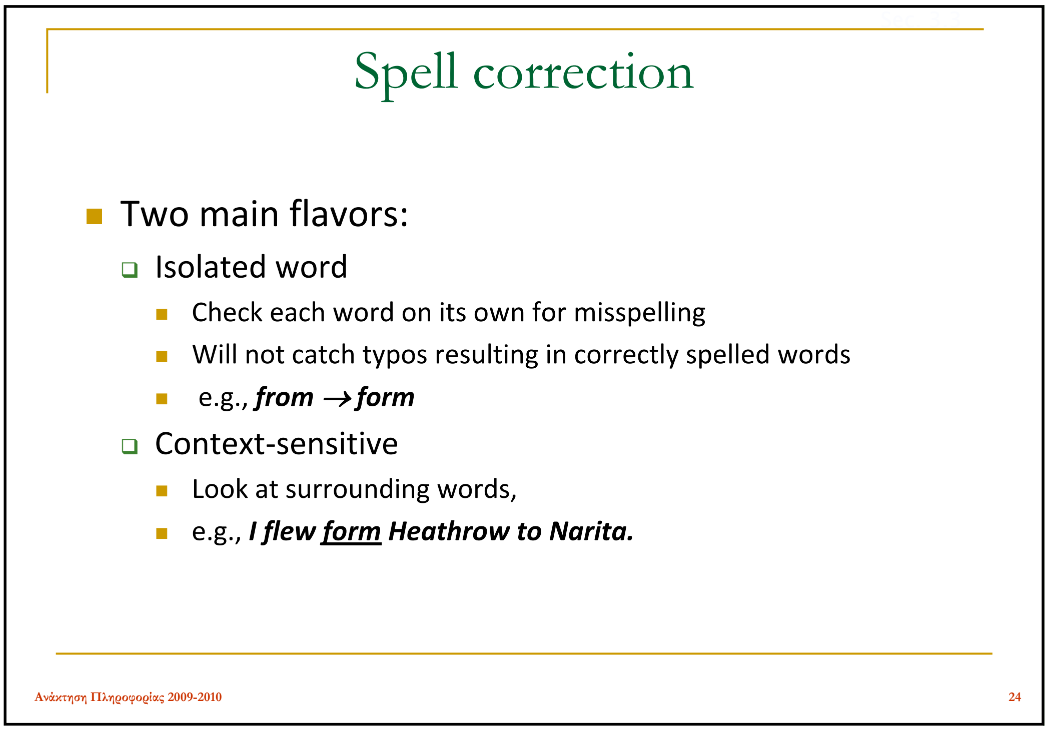
Ανάκτηση Πληροφορίας 2009-2010 22



Query mis-spellings

- Our principal focus here
  - E.g., the query ***Alanis Morisett***
- We can either
  - Retrieve documents indexed by the correct spelling, OR
  - Return several suggested alternative queries with the correct spelling
    - *Did you mean … ?*

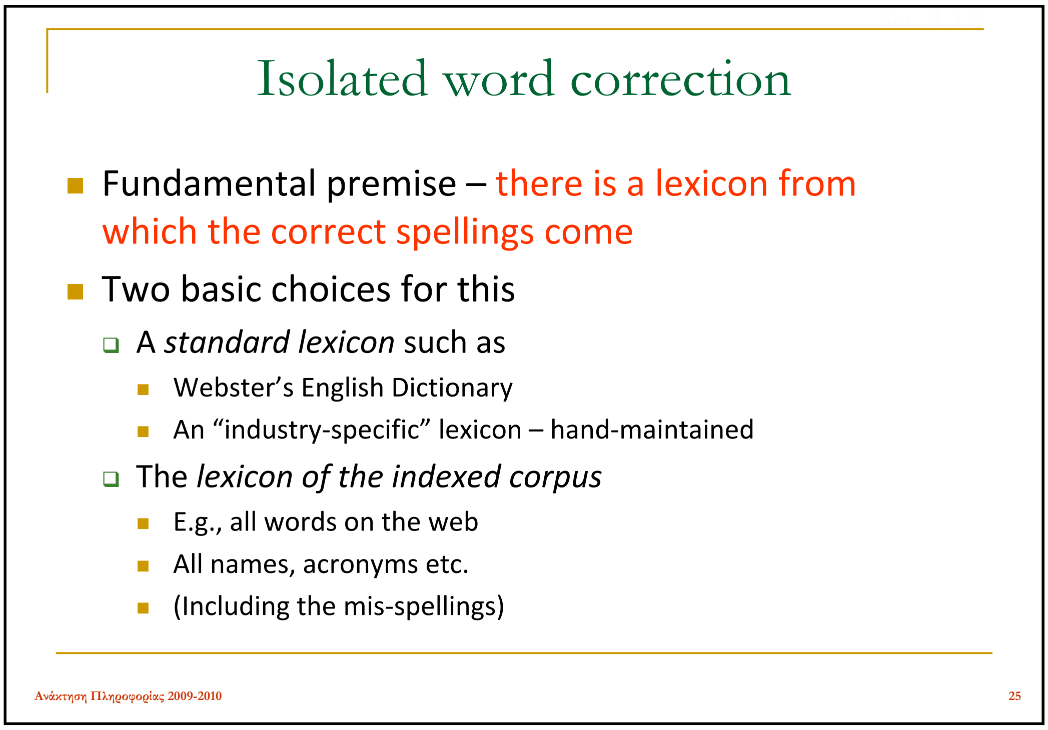
Ανάκτηση Πληροφορίας 2009-2010 23



Spell correction

- Two main flavors:
  - Isolated word
    - Check each word on its own for misspelling
    - Will not catch typos resulting in correctly spelled words
    - e.g., ***from*** $\rightarrow$ ***form***
  - Context-sensitive
    - Look at surrounding words,
    - e.g., ***I flew <u>form</u> Heathrow to Narita.***

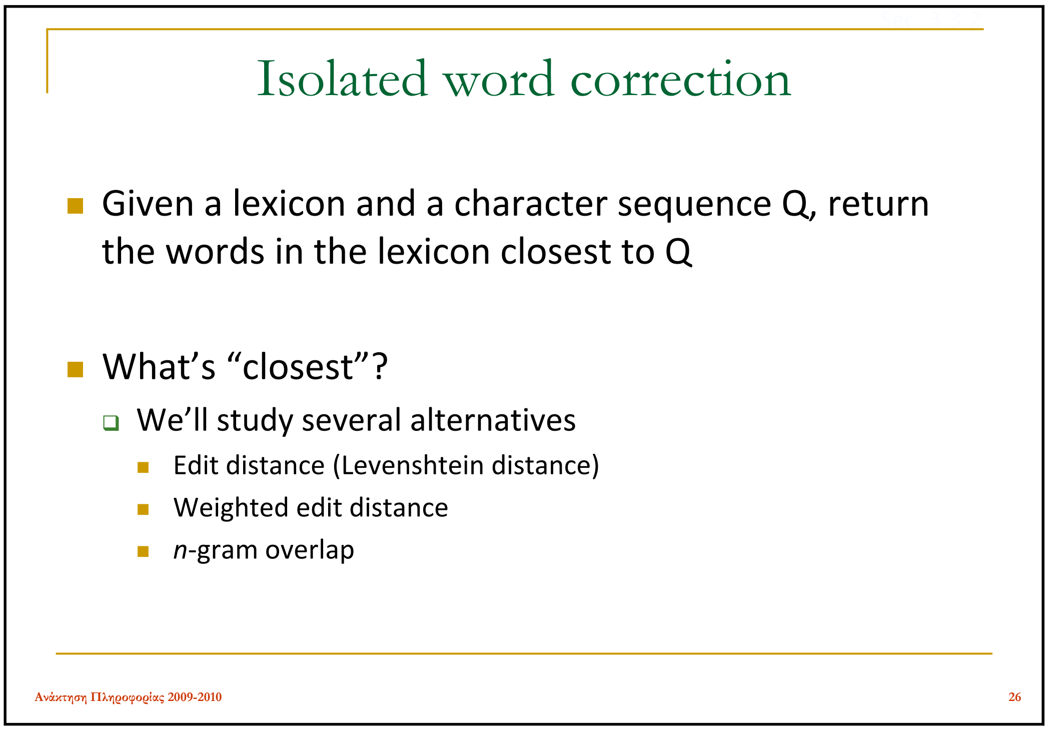
Ανάκτηση Πληροφορίας 2009-2010 24



Isolated word correction

- Fundamental premise – there is a lexicon from which the correct spellings come
- Two basic choices for this
  - A *standard lexicon* such as
    - Webster's English Dictionary
    - An "industry-specific" lexicon – hand-maintained
  - The *lexicon of the indexed corpus*
    - E.g., all words on the web
    - All names, acronyms etc.
    - (Including the mis-spellings)

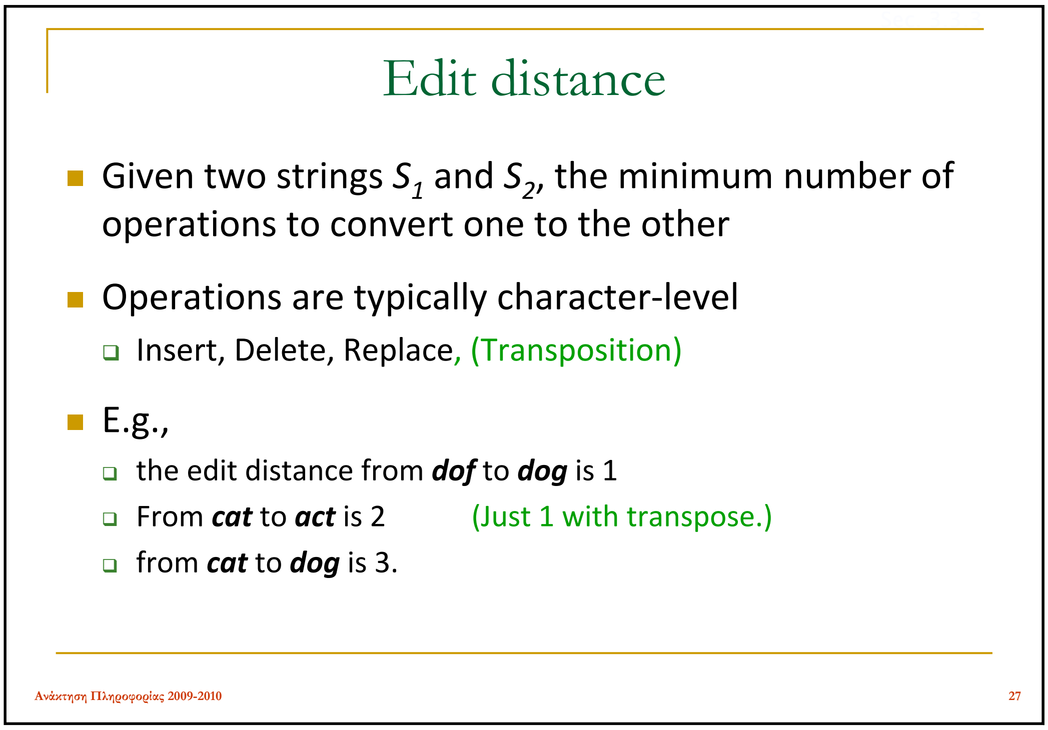
Ανάκτηση Πληροφορίας 2009-2010 25



Isolated word correction

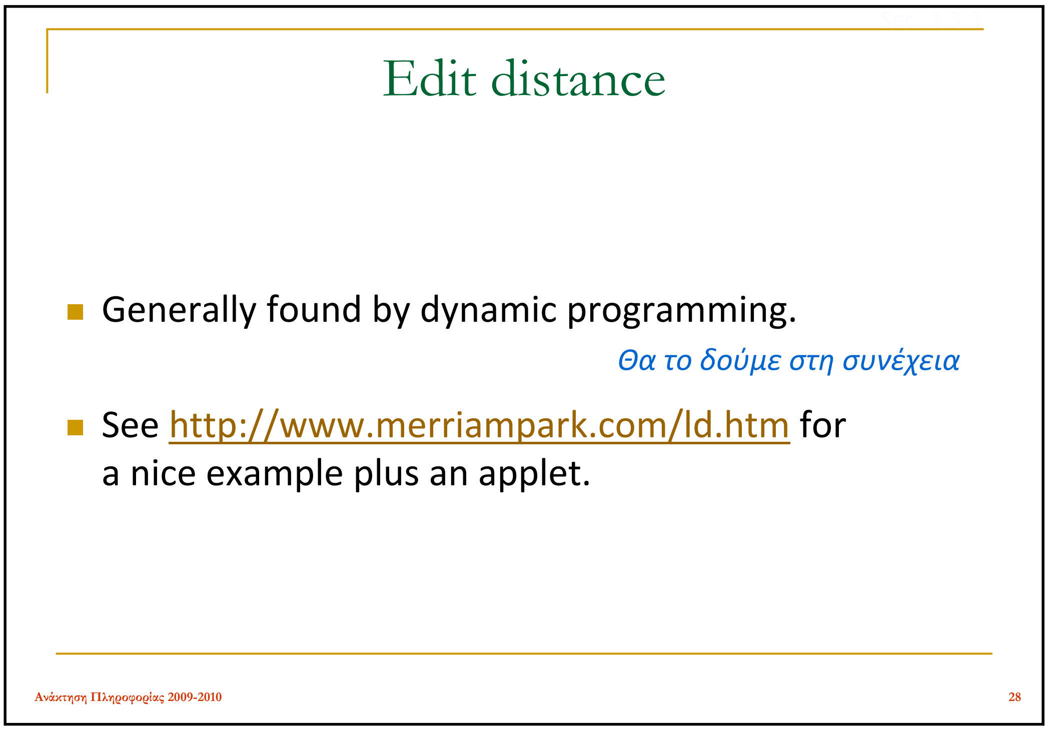
- Given a lexicon and a character sequence Q, return the words in the lexicon closest to Q
- What's "closest"?
  - We'll study several alternatives
    - Edit distance (Levenshtein distance)
    - Weighted edit distance
    - *n*-gram overlap

Ανάκτηση Πληροφορίας 2009-2010 26



Edit distance

- Given two strings S_1 and S_2, the minimum number of operations to convert one to the other
- Operations are typically character-level
  - Insert, Delete, Replace, (Transposition)
- E.g.,
  - the edit distance from ***dof*** to ***dog*** is 1
  - From ***cat*** to ***act*** is 2 (Just 1 with transpose.)
  - from ***cat*** to ***dog*** is 3.



Edit distance

- Generally found by dynamic programming.

  Θα το δούμε στη συνέχεια

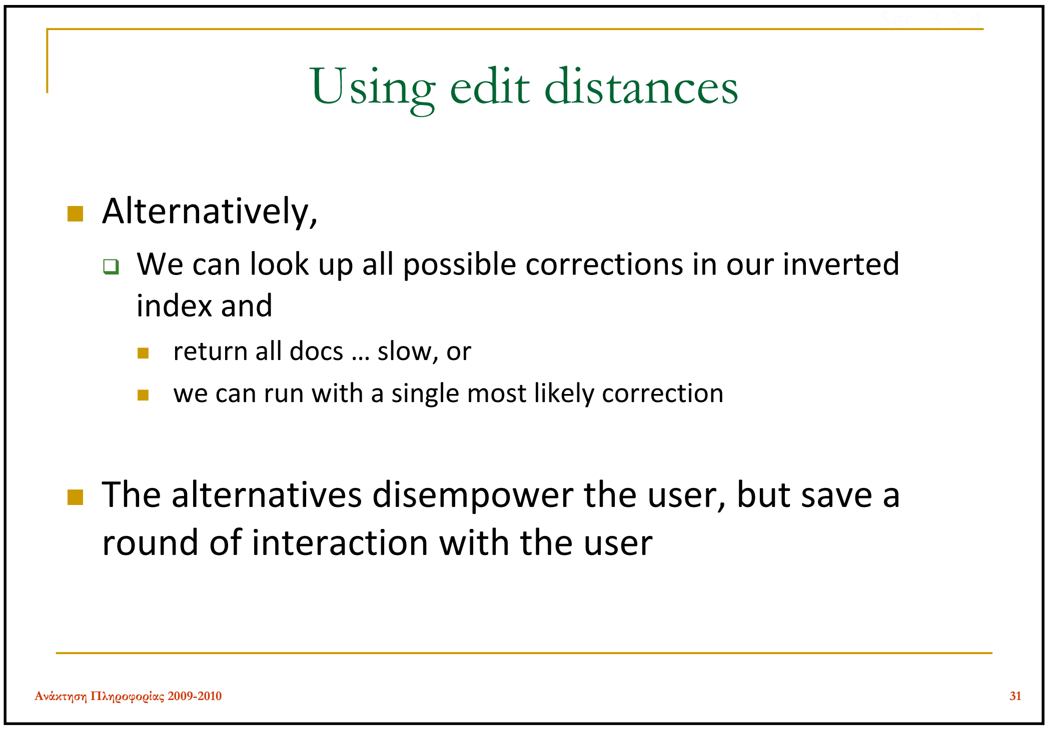
- See http://www.merriampark.com/ld.htm for a nice example plus an applet.

Weighted edit distance

- As above, but the weight of an operation depends on the character(s) involved
  - Meant to capture OCR or keyboard errors, e.g. ***m*** more likely to be mis-typed as ***n*** than as ***q***
  - Therefore, replacing ***m*** by ***n*** is a smaller edit distance than by ***q***
  - This may be formulated as a probability model
- Requires weight matrix as input
- Modify dynamic programming to handle weights

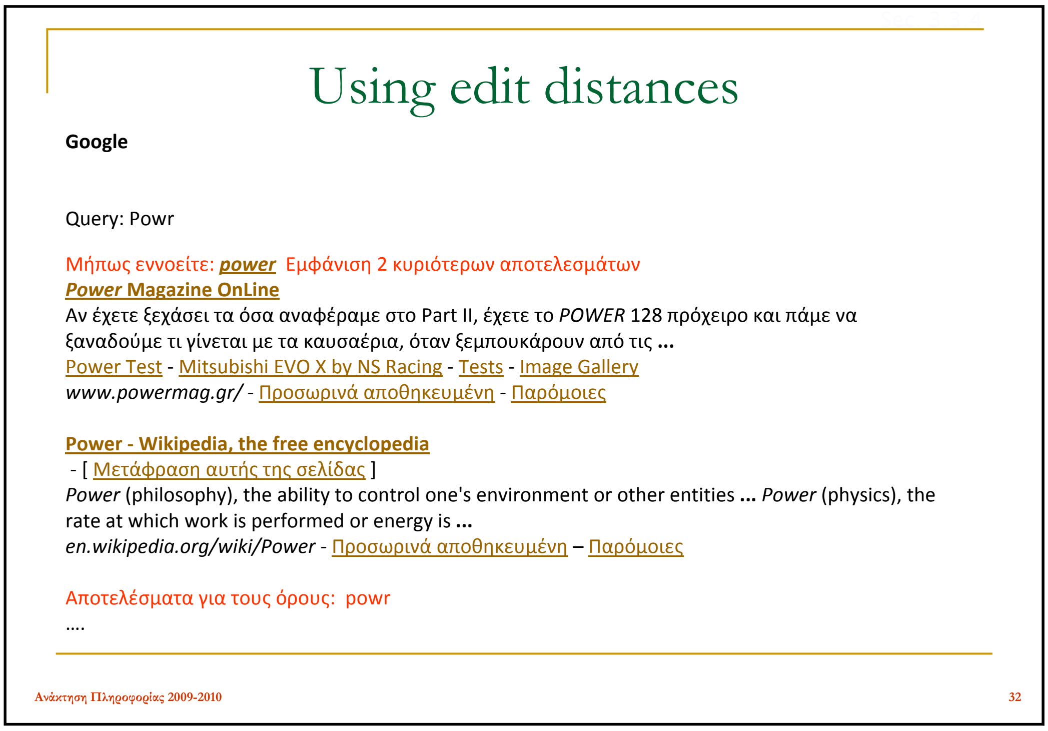
Using edit distances

1. Given query, first *enumerate* all character sequences within a preset (weighted) edit distance (e.g., 2)
2. Intersect this set with list of “correct” words
3. Show terms you found to user as *suggestions*



Using edit distances

- Alternatively,
  - We can look up all possible corrections in our inverted index and
    - return all docs … slow, or
    - we can run with a single most likely correction
- The alternatives disempower the user, but save a round of interaction with the user



Using edit distances

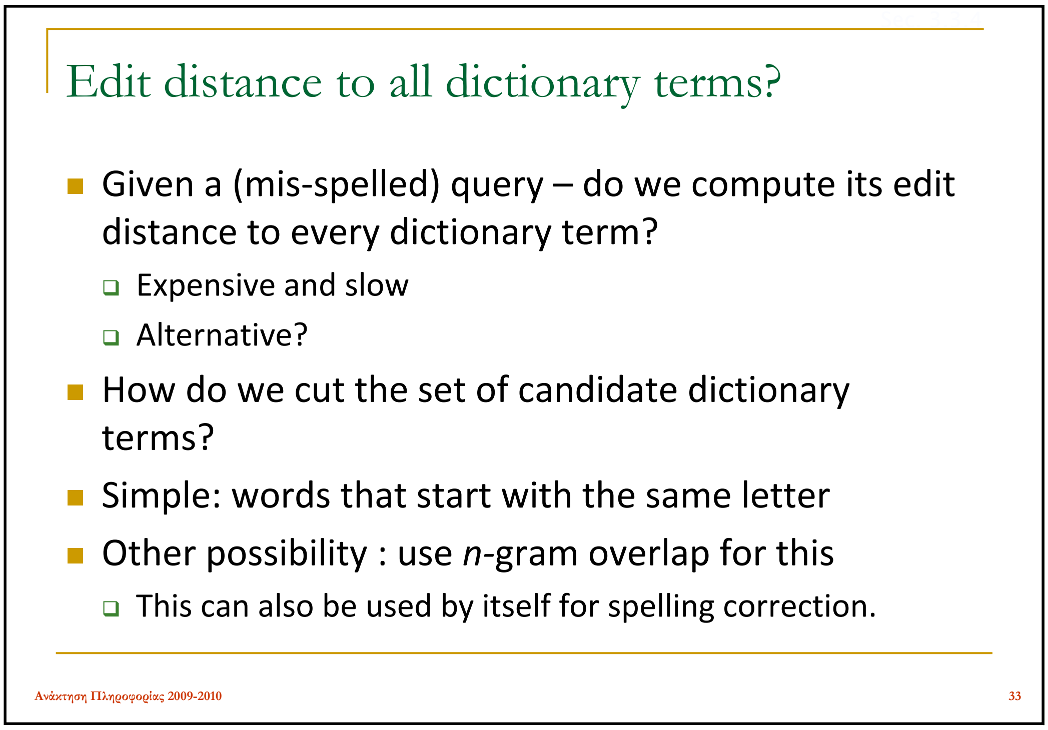
Google

Query: Powr

Μήπως εννοείτε: ***power*** Εμφάνιση 2 κυριότερων αποτελεσμάτων
***Power* Magazine OnLine**
Αν έχετε ξεχάσει τα όσα αναφέραμε στο Part II, έχετε το *POWER* 128 πρόχειρο και πάμε να ξαναδούμε τι γίνεται με τα καυσαέρια, όταν ξεμπουκάρουν από τις **...**
Power Test - Mitsubishi EVO X by NS Racing - Tests - Image Gallery
www.powermag.gr/ - Προσωρινά αποθηκευμένη - Παρόμοιες

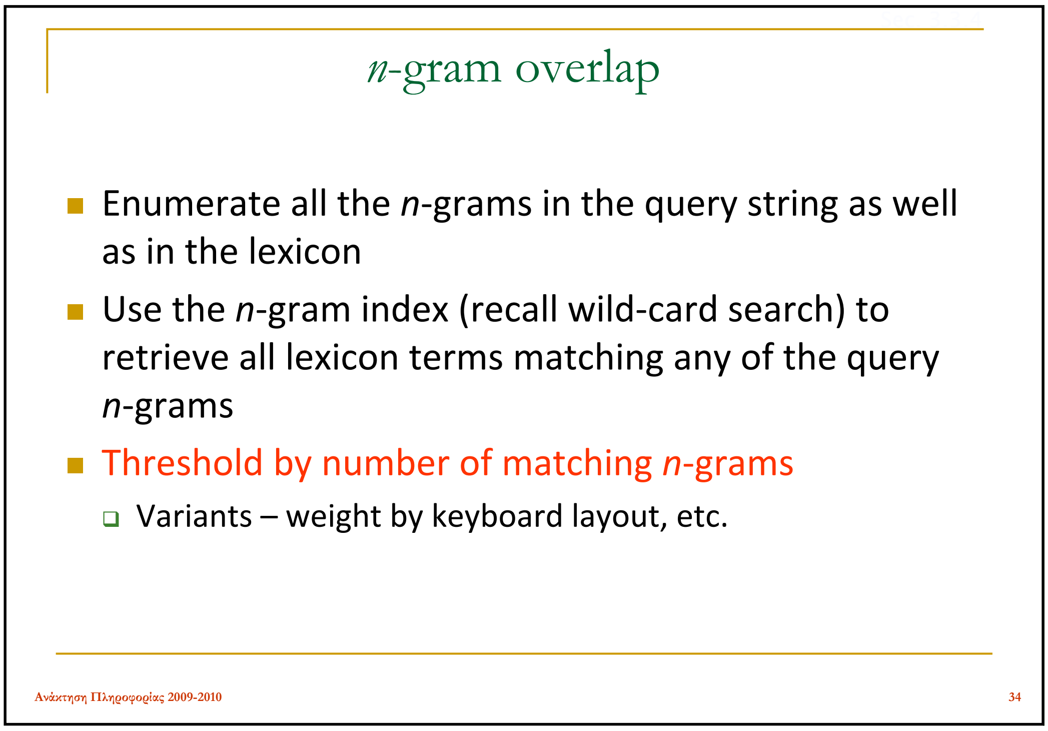
Power - Wikipedia, the free encyclopedia
 - [Μετάφραση αυτής της σελίδας]
Power (philosophy), the ability to control one's environment or other entities **...** *Power* (physics), the rate at which work is performed or energy is **...**
en.wikipedia.org/wiki/Power - Προσωρινά αποθηκευμένη – Παρόμοιες

Αποτελέσματα για τους όρους: powr
....



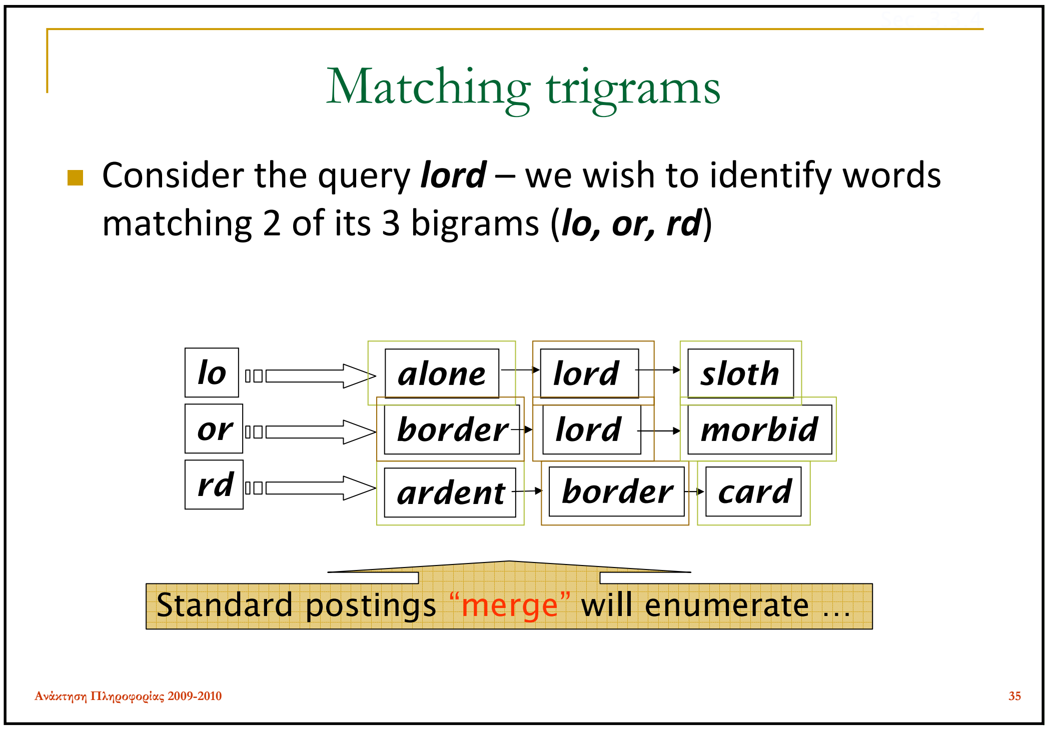
Edit distance to all dictionary terms?

- Given a (mis-spelled) query – do we compute its edit distance to every dictionary term?
  - Expensive and slow
  - Alternative?
- How do we cut the set of candidate dictionary terms?
- Simple: words that start with the same letter
- Other possibility : use *n*-gram overlap for this
  - This can also be used by itself for spelling correction.



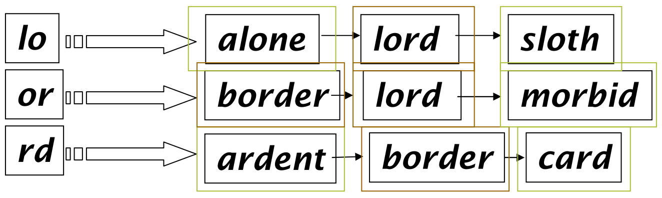
n-gram overlap

- Enumerate all the *n*-grams in the query string as well as in the lexicon
- Use the *n*-gram index (recall wild-card search) to retrieve all lexicon terms matching any of the query *n*-grams
- Threshold by number of matching *n*-grams
  - Variants – weight by keyboard layout, etc.

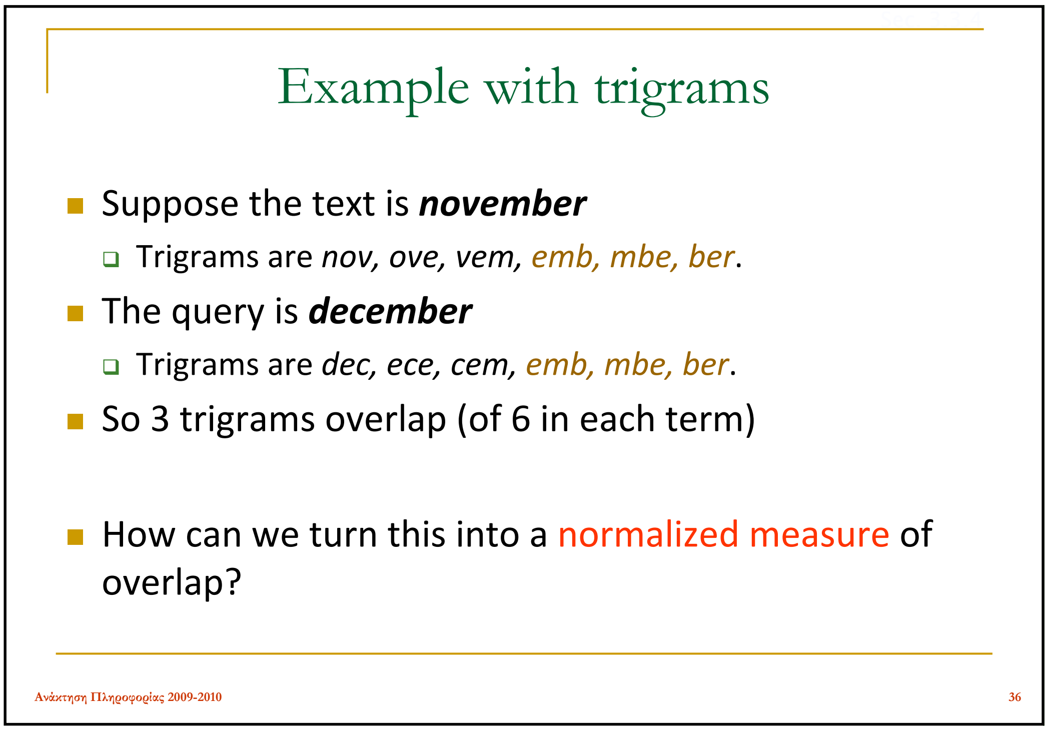


Matching trigrams

- Consider the query ***lord*** – we wish to identify words matching 2 of its 3 bigrams (***lo, or, rd***)

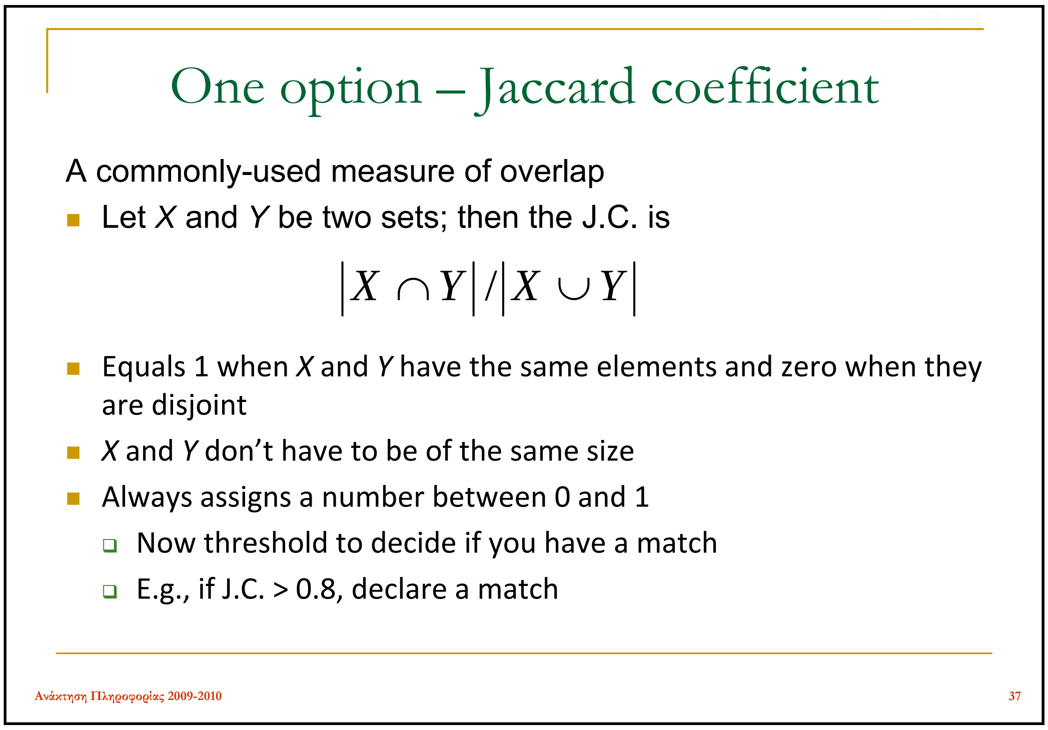


Standard postings “merge” will enumerate …



Example with trigrams

- Suppose the text is ***november***
  - Trigrams are *nov, ove, vem, emb, mbe, ber*.
- The query is ***december***
  - Trigrams are *dec, ece, cem, emb, mbe, ber*.
- So 3 trigrams overlap (of 6 in each term)
- How can we turn this into a normalized measure of overlap?



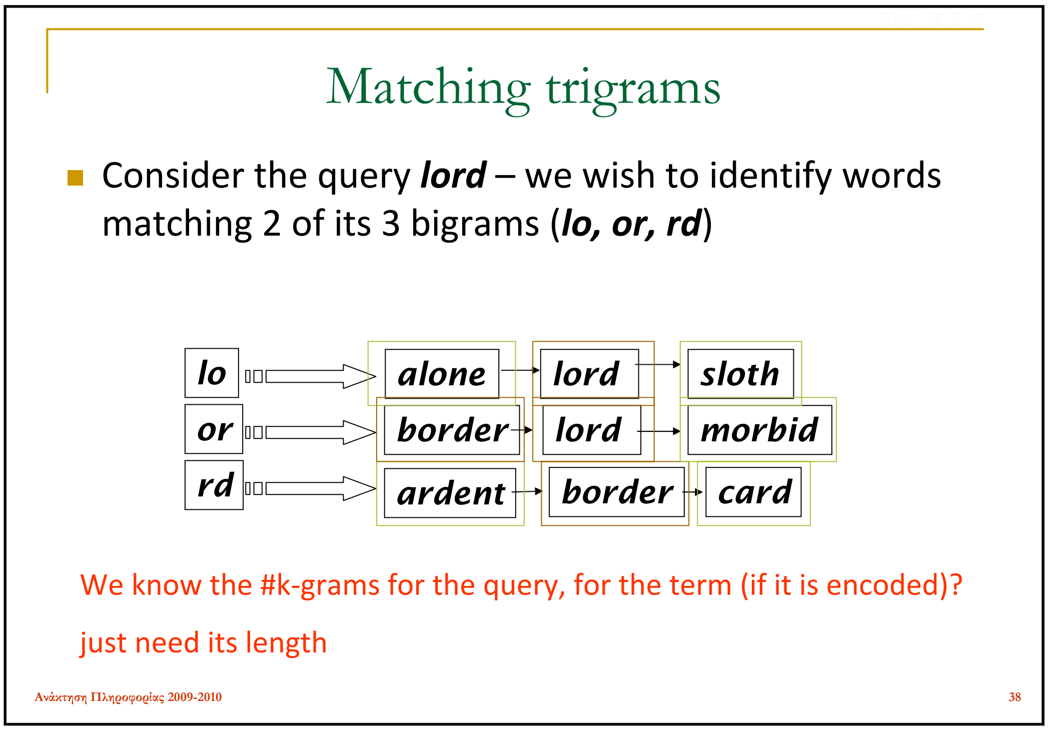
One option – Jaccard coefficient

A commonly-used measure of overlap

- Let *X* and *Y* be two sets; then the J.C. is

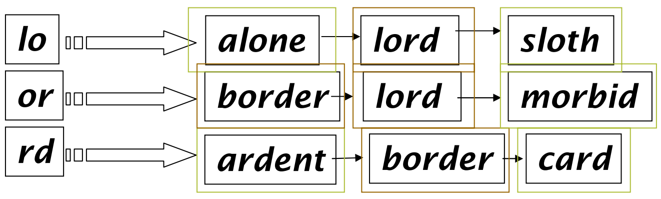
$$|X \cap Y| / |X \cup Y|$$

- Equals 1 when *X* and *Y* have the same elements and zero when they are disjoint
- *X* and *Y* don't have to be of the same size
- Always assigns a number between 0 and 1
  - Now threshold to decide if you have a match
  - E.g., if J.C. > 0.8, declare a match



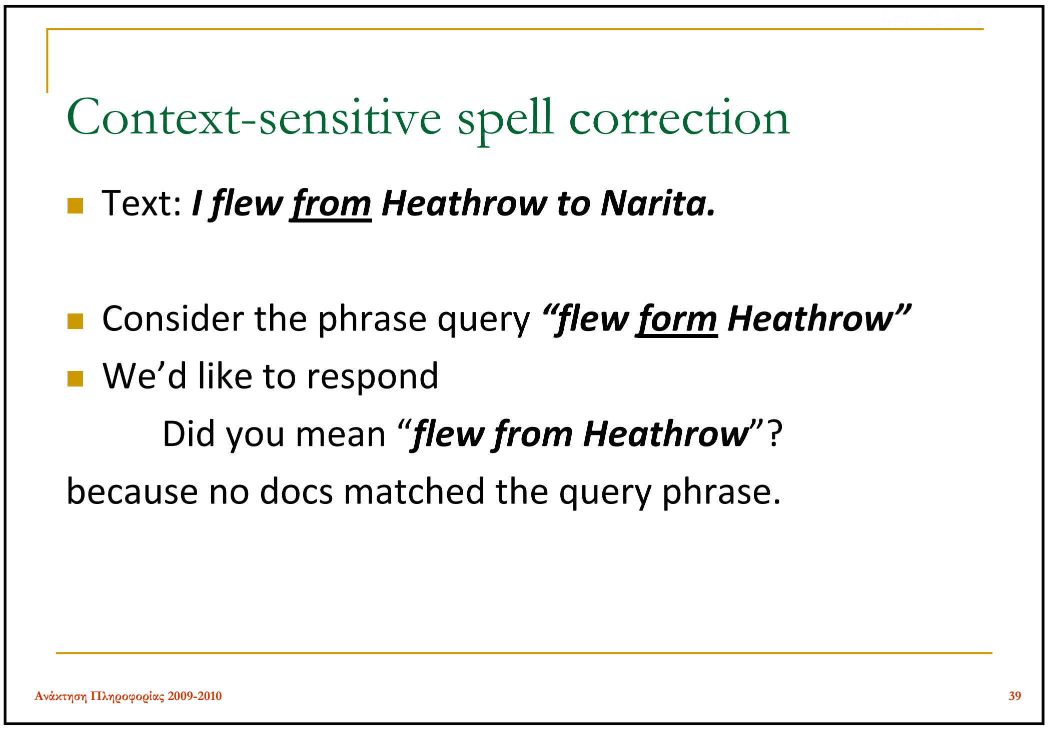
Matching trigrams

- Consider the query ***lord*** – we wish to identify words matching 2 of its 3 bigrams (***lo, or, rd***)



We know the #k-grams for the query, for the term (if it is encoded)?

just need its length



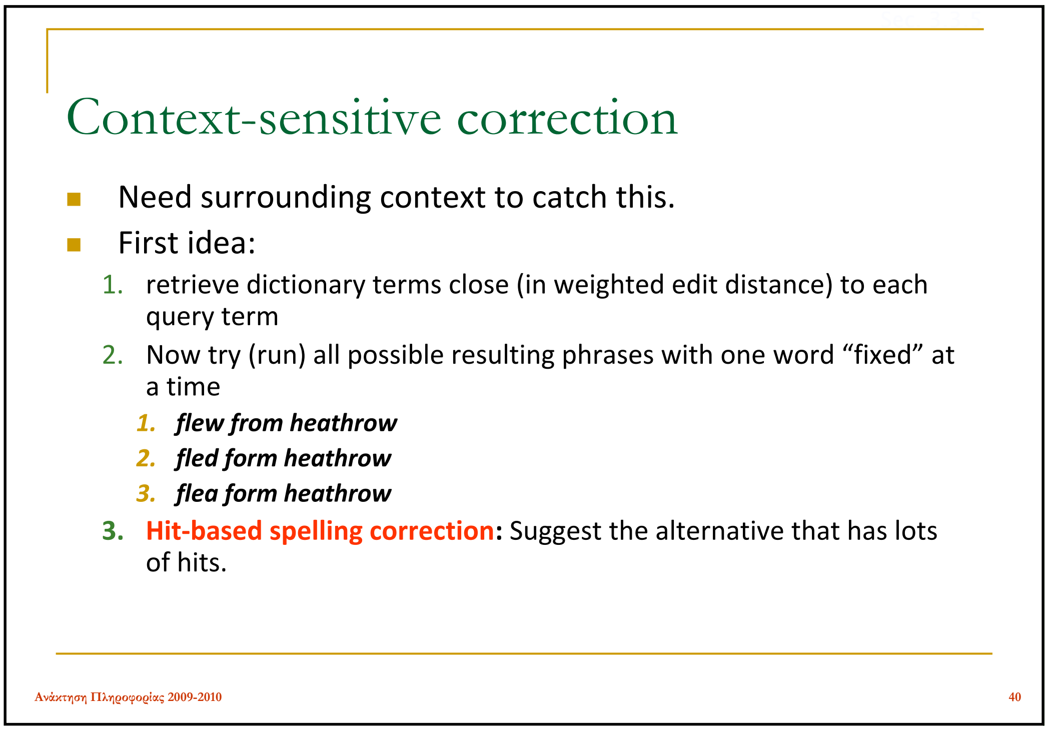
Context-sensitive spell correction

- Text: ***I flew from Heathrow to Narita.***

- Consider the phrase query ***"flew form Heathrow"***
- We'd like to respond

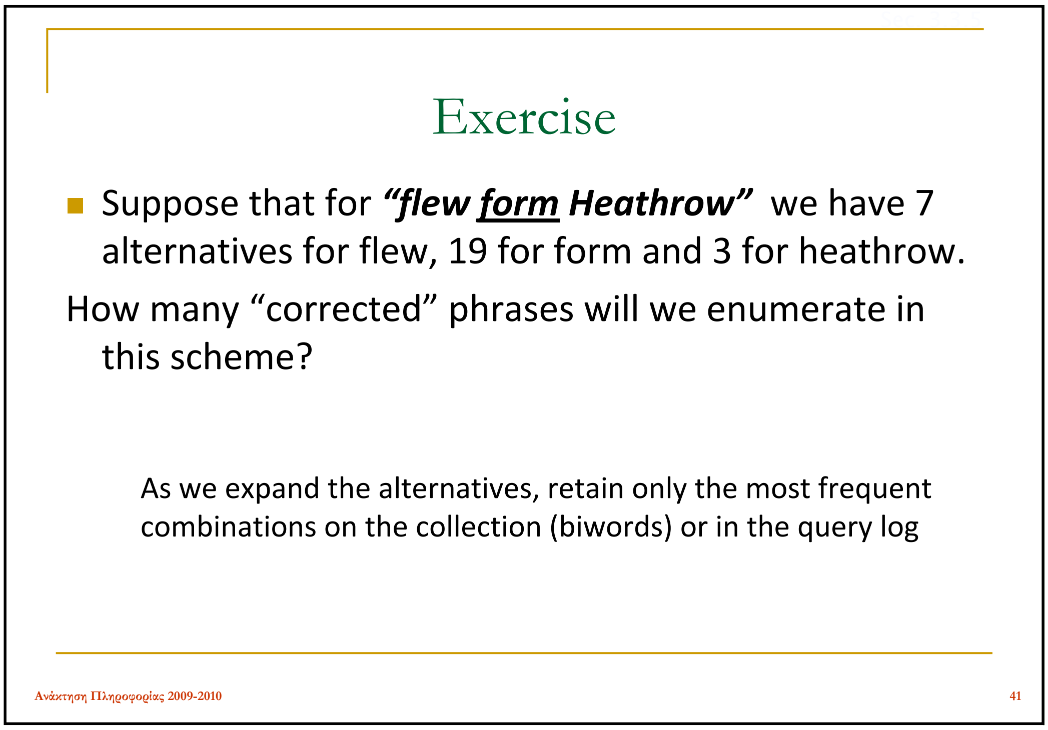
  Did you mean "***flew from Heathrow***"?

because no docs matched the query phrase.



Context-sensitive correction

- Need surrounding context to catch this.
- First idea:
  1. retrieve dictionary terms close (in weighted edit distance) to each query term
  2. Now try (run) all possible resulting phrases with one word "fixed" at a time
     1. ***flew from heathrow***
     2. ***fled form heathrow***
     3. ***flea form heathrow***
  3. **Hit-based spelling correction:** Suggest the alternative that has lots of hits.



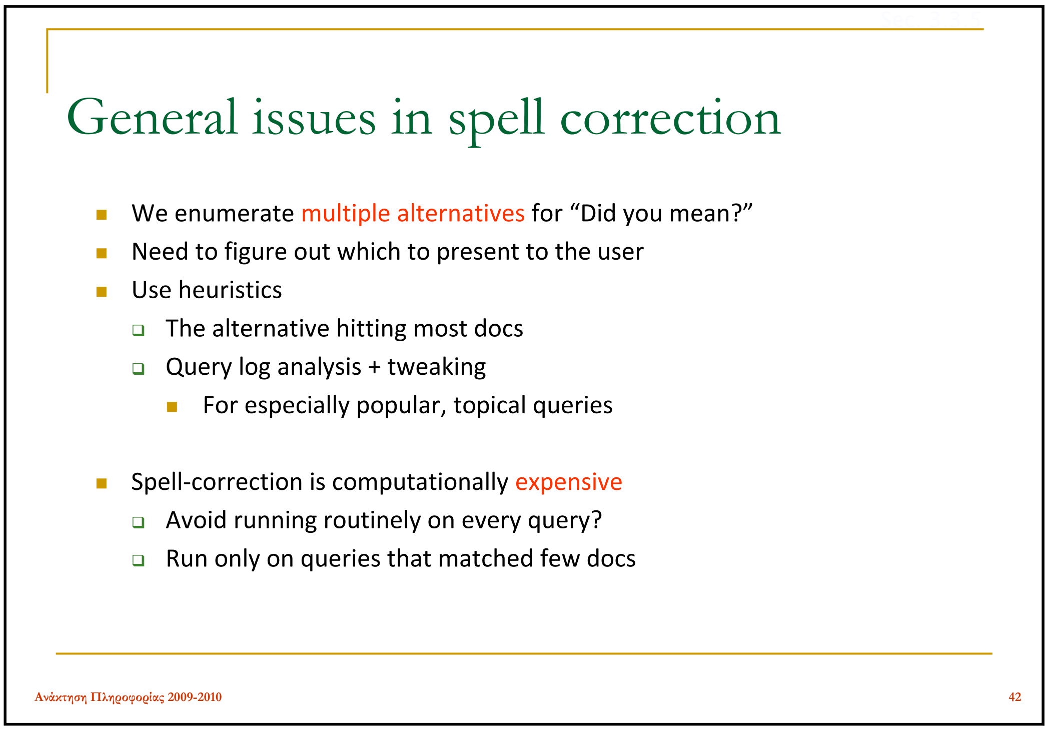
Exercise

- Suppose that for ***"flew <u>form</u> Heathrow"*** we have 7 alternatives for flew, 19 for form and 3 for heathrow.

How many "corrected" phrases will we enumerate in this scheme?

As we expand the alternatives, retain only the most frequent combinations on the collection (biwords) or in the query log

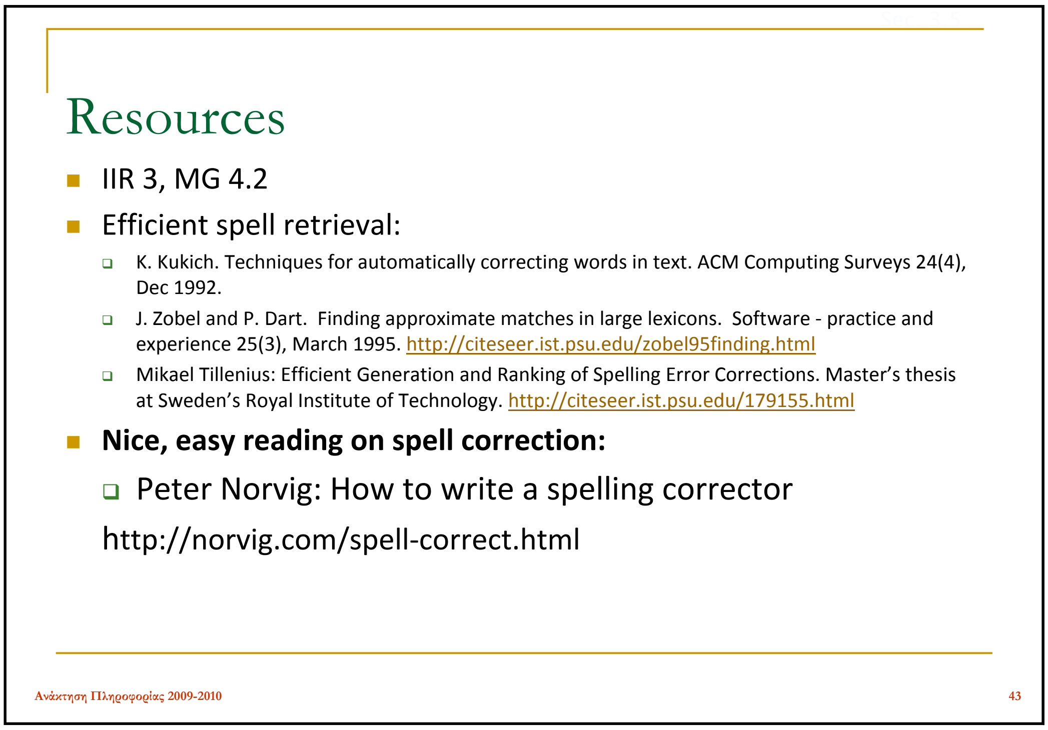
Ανάκτηση Πληροφορίας 2009-2010 41



General issues in spell correction

- We enumerate multiple alternatives for "Did you mean?"
- Need to figure out which to present to the user
- Use heuristics
  - The alternative hitting most docs
  - Query log analysis + tweaking
    - For especially popular, topical queries
- Spell-correction is computationally expensive
  - Avoid running routinely on every query?
  - Run only on queries that matched few docs

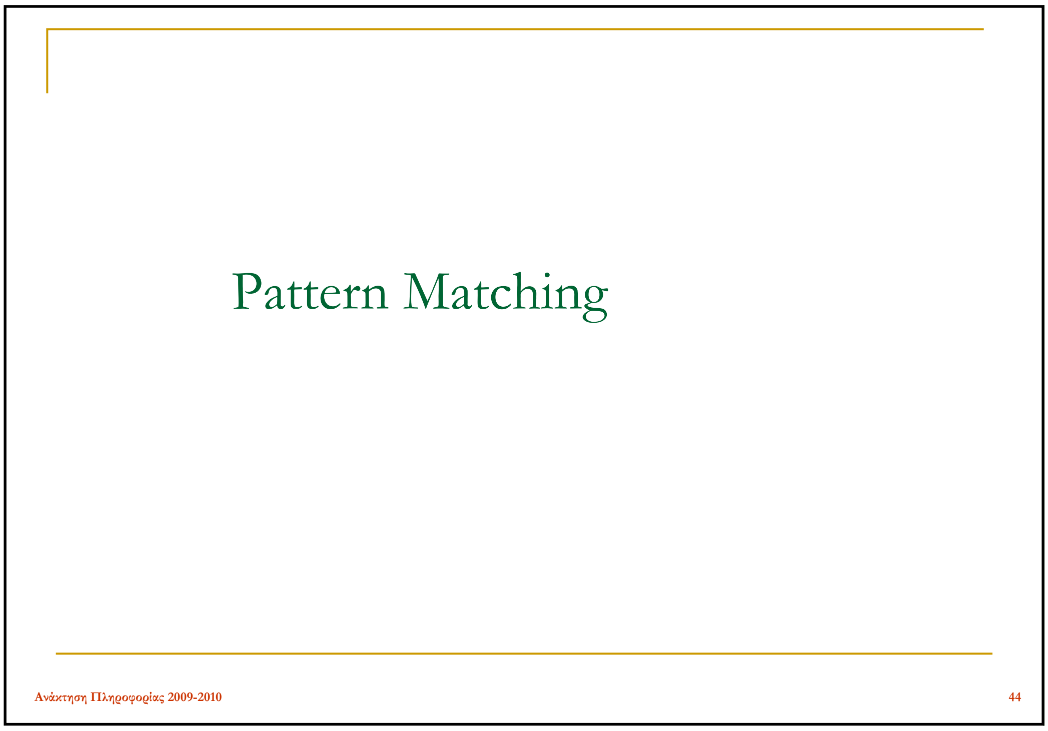
Ανάκτηση Πληροφορίας 2009-2010 42



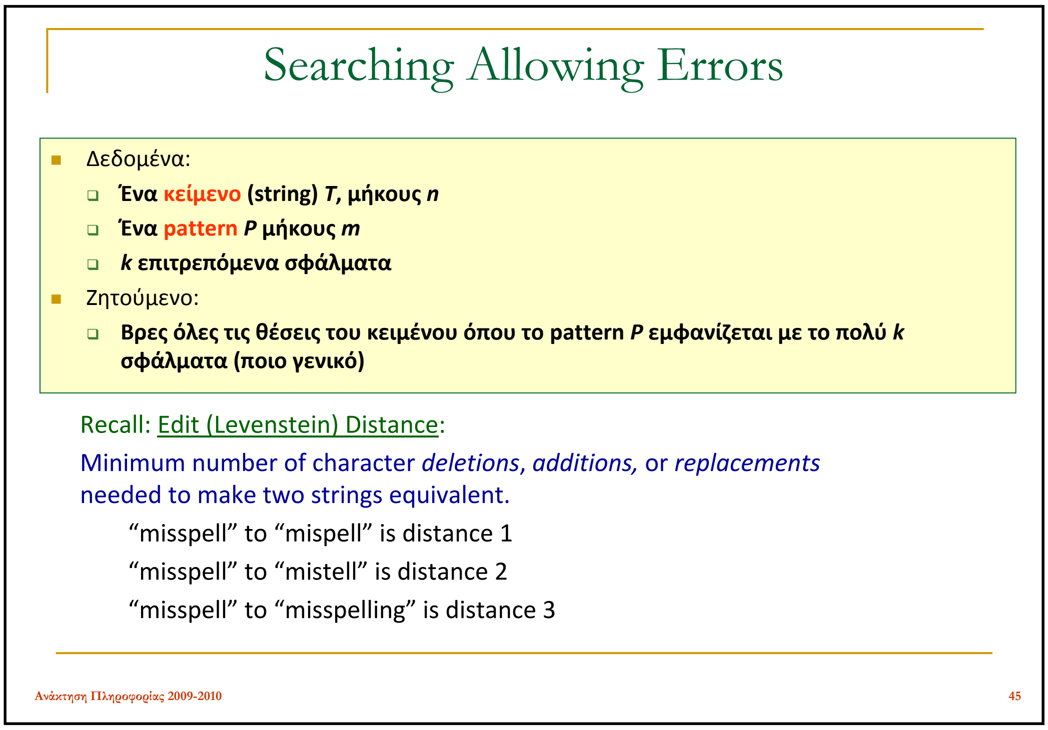
Resources

- IIR 3, MG 4.2
- Efficient spell retrieval:
  - K. Kukich. Techniques for automatically correcting words in text. ACM Computing Surveys 24(4), Dec 1992.
  - J. Zobel and P. Dart. Finding approximate matches in large lexicons. Software - practice and experience 25(3), March 1995. http://citeseer.ist.psu.edu/zobel95finding.html
  - Mikael Tillenius: Efficient Generation and Ranking of Spelling Error Corrections. Master's thesis at Sweden's Royal Institute of Technology. http://citeseer.ist.psu.edu/179155.html
- **Nice, easy reading on spell correction:**
  - Peter Norvig: How to write a spelling corrector

  http://norvig.com/spell-correct.html



Pattern Matching



Searching Allowing Errors

- Δεδομένα:
  - **Ένα κείμενο (string) *T*, μήκους *n***
  - **Ένα pattern *P* μήκους *m***
  - ***k* επιτρεπόμενα σφάλματα**
- Ζητούμενο:
  - **Βρες όλες τις θέσεις του κειμένου όπου το pattern *P* εμφανίζεται με το πολύ *k* σφάλματα (ποιο γενικό)**

Recall: Edit (Levenstein) Distance:

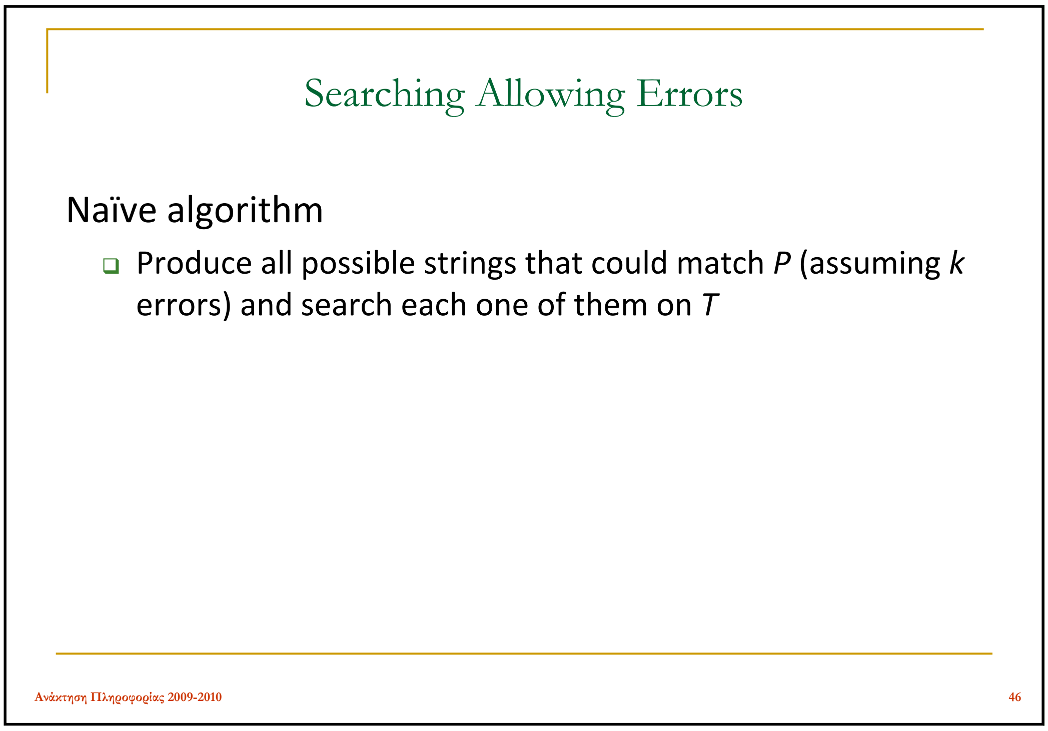
Minimum number of character *deletions*, *additions,* or *replacements* needed to make two strings equivalent.

"misspell" to "mispell" is distance 1

"misspell" to "mistell" is distance 2

"misspell" to "misspelling" is distance 3

Ανάκτηση Πληροφορίας 2009-2010 45



Searching Allowing Errors

Naïve algorithm

- Produce all possible strings that could match *P* (assuming *k* errors) and search each one of them on *T*

Ανάκτηση Πληροφορίας 2009-2010 46

Searching Allowing Errors:
Solution using **Dynamic Programming**

- Dynamic Programming is the class of algorithms, which includes the most commonly used algorithms in speech and language processing.
- Among them the **minimum edit distance algorithm for spelling error correction.**
- Intuition:
  - *a large problem can be solved by properly combining the solutions to various subproblems.*

Searching Allowing Errors:
Solution using **Dynamic Programming**

Έναν m x n πίνακα C

Γραμμές θέσεις του pattern

Στήλες θέσεις του text

C[i, j]: ο ελάχιστος αριθμός λαθών για να ταιριάξουμε το $P_{1..i}$ με ένα sufix του $T_{1..j}$

C[0, j] = 0

C[i, 0] = i /* delete i characters

Η ιδέα είναι ο υπολογισμός μιας τιμής του πίνακα με βάση τις προηγούμενες (δηλαδή, ήδη υπολογισμένες) γειτονικές της

Searching Allowing Errors: Solution using **Dynamic Programming**

C[i, j]: ο ελάχιστος αριθμός λαθών για να ταιριάξουμε το $P_{1..i}$ με ένα suffix του $T_{1..j}$

```
C[i, j] =
αν $P_i = T_j$
    τότε C[i-1, j-1]
Αλλιώς ο καλύτερος τρόπος από τα παρακάτω
            replace $P_i$ με $T_j$ (ή το συμμετρικό) κόστος 1+C[i-1,j-1]
            delete $P_i$ κόστος 1+ C[i-l, j]
            delete $T_j$ 1 + C[i,j-1]
            add ??
```

Searching Allowing Errors: Solution using **Dynamic Programming (II)**

Problem Statement: T[n] text string, P[m] pattern, k errors

Example: T = “surgery”, P = “survey”, k=**2**

To explain the algorithm we will use a m x n matrix C

one row for each char of P, one column for each char of T

(latter on we shall see that we need less space)

T

P

		s	u	r	g	e	r	y
s								
u								
r								
v								
e								
y								

Searching Allowing Errors: Solution using **Dynamic Programming (III)**

T = “surgery”, P = “survey”, k=2

οι γραμμές του C εκφράζουν πόσα γράμματα του pattern έχουμε ήδη καταναλώσει (στη 0-γραμμή τίποτα, στη m-γραμμή ολόκληρο το pattern)

C[0,j] := 0 for every column j

(no letter of P has been consumed)

C[i,0] := i for every row i

(i chars of P have been consumed, pointer of T at 0. So i errors (insertions) so far)

T

P

		s	u	r	g	e	r	y
	0	0	0	0	0	0	0	0
s	1							
u	2							
r	3							
v	4							
e	5							
y	6							

Searching Allowing Errors: Solution using **Dynamic Programming (IV)**

if P[i]=T[j] **THEN** C[i,j] := C[i-1,j-1]

// εγινε match άρα τα “λάθη” ήταν όσα και πριν

Else C[i,j] := **1** + *min* of:

- C[i-1,j]
  - // i-1 chars consumed P, j chars consumed of T
  - // ~delete a char from T
- C[i,j-1]
  - // i chars consumed P, j-1 chars consumed of T
  - // ~ delete a char from P
- C[i-1,j-1]
  - // i-1 chars consumed P, j-1 chars consumed of T
  - // ~ character replacement

Searching Allowing Errors: Solution using **Dynamic Programming: Example**

T = “surgery”, P = “survey”, k=2

T

P

		s	u	r	g	e	r	y
	0	0	0	0	0	0	0	0
s	1	0	1	1	1	1	1	1
u	2	1	0	1	2	2	2	2
r	3	2	1	0	1	2	2	3
v	4	3	2	1	1	2	3	3
e	5	4	3	2	2	1	2	3
y	6	5	4	3	3	**2**	**2**	**2**

Solution using **Dynamic Programming: Example**

T = “surgery”, P = “survey”, k=2

T

P

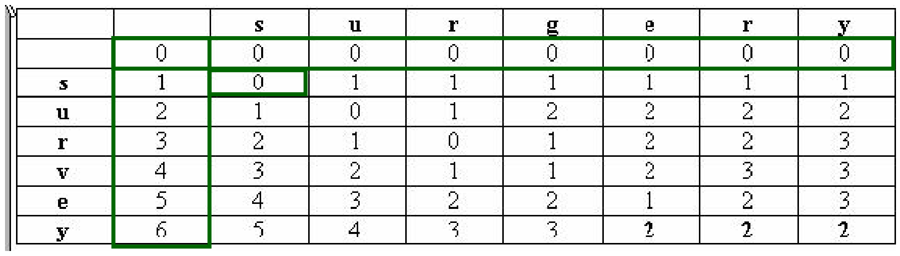
		s	u	r	g	e	r	y
	0	0	0	0	0	0	0	0
s	1	0	1	1	1	1	1	1
u	2	1	0	1	2	2	2	2
r	3	2	1	0	1	2	2	3
v	4	3	2	1	1	2	3	3
e	5	4	3	2	2	1	2	3
y	6	5	4	3	3	**2**	**2**	**2**

Solution using **Dynamic Programming: Example**

T = “surgery”, P = “survey”, k=2

T

P



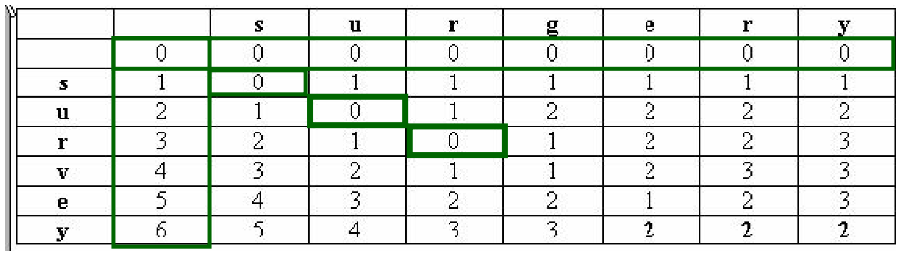
		s	u	r	g	e	r	y
	0	0	0	0	0	0	0	0
s	1	0	1	1	1	1	1	1
u	2	1	0	1	2	2	2	2
r	3	2	1	0	1	2	2	3
v	4	3	2	1	1	2	3	3
e	5	4	3	2	2	1	2	3
y	6	5	4	3	3	2	2	2

Solution using **Dynamic Programming: Example**

T = “surgery”, P = “survey”, k=2

T

P



		s	u	r	g	e	r	y
	0	0	0	0	0	0	0	0
s	1	0	1	1	1	1	1	1
u	2	1	0	1	2	2	2	2
r	3	2	1	0	1	2	2	3
v	4	3	2	1	1	2	3	3
e	5	4	3	2	2	1	2	3
y	6	5	4	3	3	2	2	2

Solution using **Dynamic Programming: Example**

T = “surgery”, P = “survey”, k=2

T

P

		s	u	r	g	e	r	y
	0	0	0	0	0	0	0	0
s	1	0	1	1	1	1	1	1
u	2	1	0	1	2	2	2	2
r	3	2	1	0	1	2	2	3
v	4	3	2	1	1	2	3	3
e	5	4	3	2	2	1	2	3
y	6	5	4	3	3	2	2	2

1 + $\boxed{\quad\quad}$

Solution using **Dynamic Programming: Example**

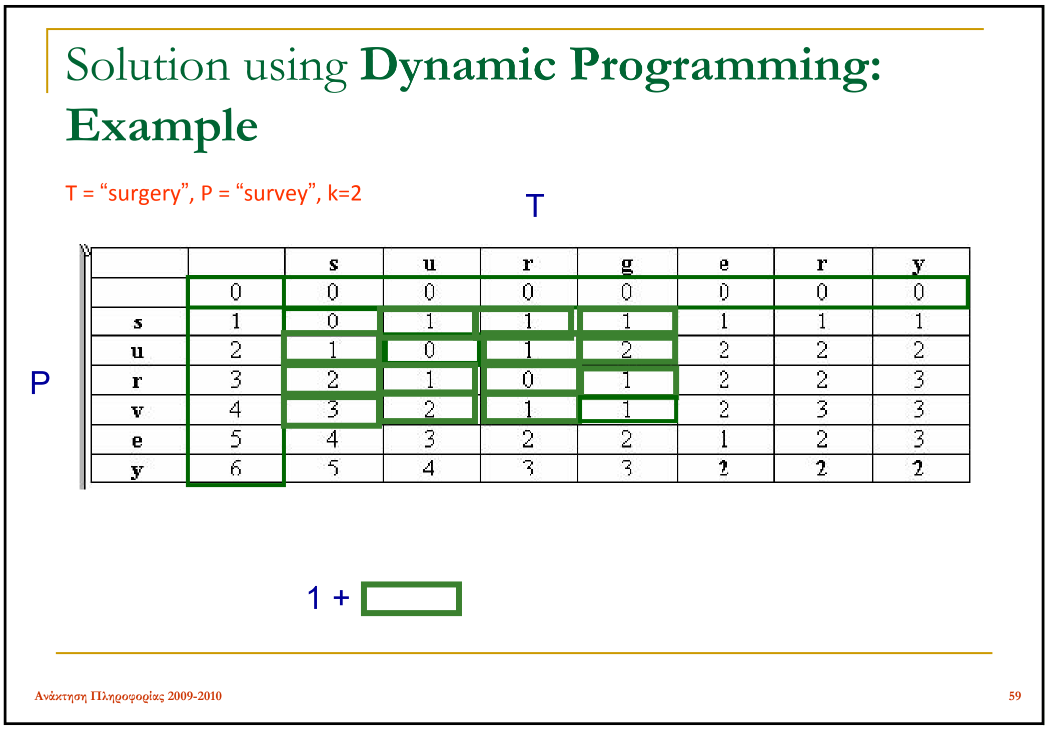
T = “surgery”, P = “survey”, k=2

T

P

		s	u	r	g	e	r	y
	0	0	0	0	0	0	0	0
s	1	0	1	1	1	1	1	1
u	2	1	0	1	2	2	2	2
r	3	2	1	0	1	2	2	3
v	4	3	2	1	1	2	3	3
e	5	4	3	2	2	1	2	3
y	6	5	4	3	3	2	2	2

1 + $\boxed{\quad\quad}$



Solution using **Dynamic Programming: Example**

T = “surgery”, P = “survey”, k=2

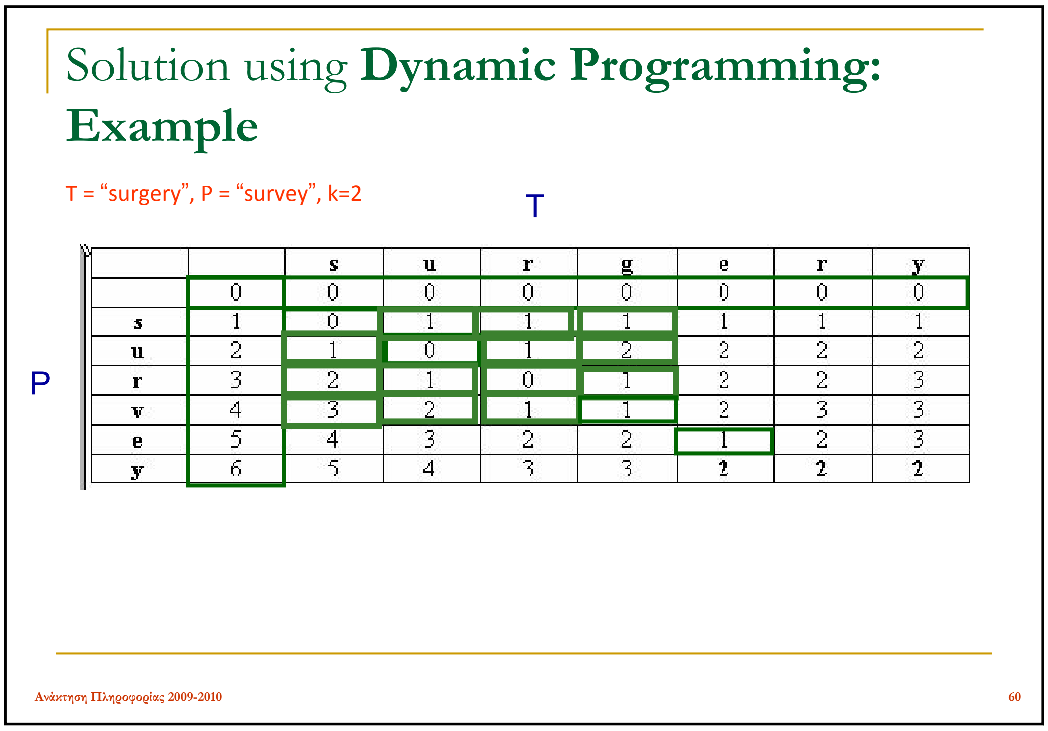
T

P

		s	u	r	g	e	r	y
	0	0	0	0	0	0	0	0
s	1	0	1	1	1	1	1	1
u	2	1	0	1	2	2	2	2
r	3	2	1	0	1	2	2	3
v	4	3	2	1	1	2	3	3
e	5	4	3	2	2	1	2	3
y	6	5	4	3	3	2	2	2

1 + $\boxed{\quad\quad}$

Ανάκτηση Πληροφορίας 2009-2010 59



Solution using **Dynamic Programming: Example**

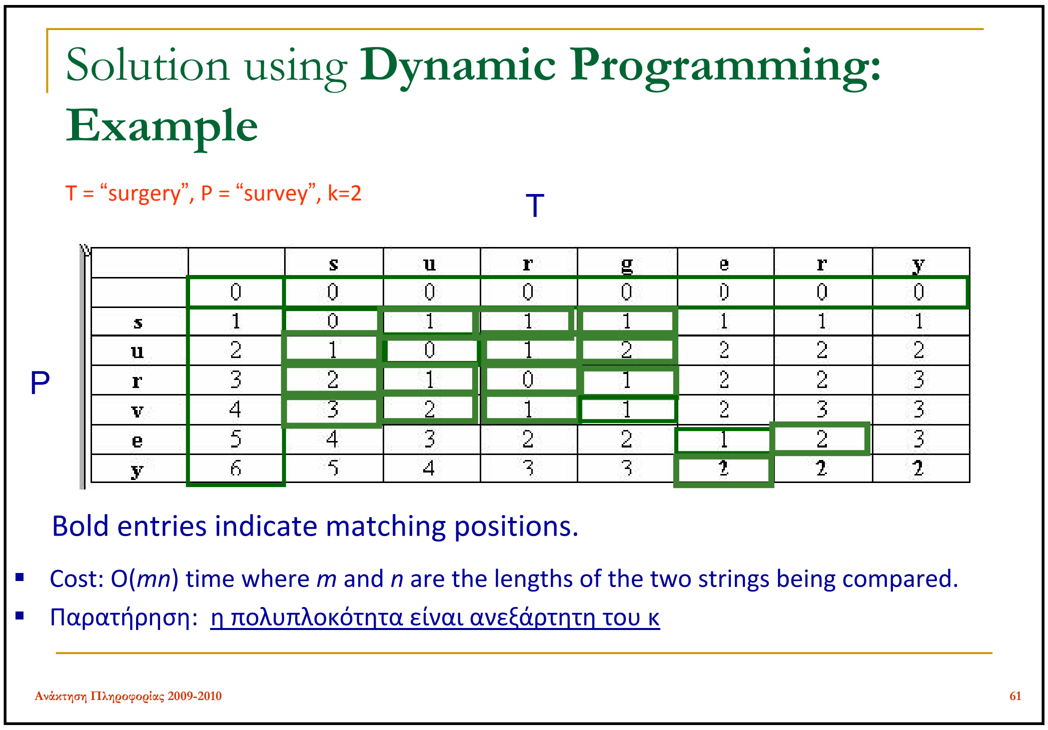
T = “surgery”, P = “survey”, k=2

T

P

		s	u	r	g	e	r	y
	0	0	0	0	0	0	0	0
s	1	0	1	1	1	1	1	1
u	2	1	0	1	2	2	2	2
r	3	2	1	0	1	2	2	3
v	4	3	2	1	1	2	3	3
e	5	4	3	2	2	1	2	3
y	6	5	4	3	3	2	2	2

Ανάκτηση Πληροφορίας 2009-2010 60



Solution using **Dynamic Programming: Example**

T = “surgery”, P = “survey”, k=2

T

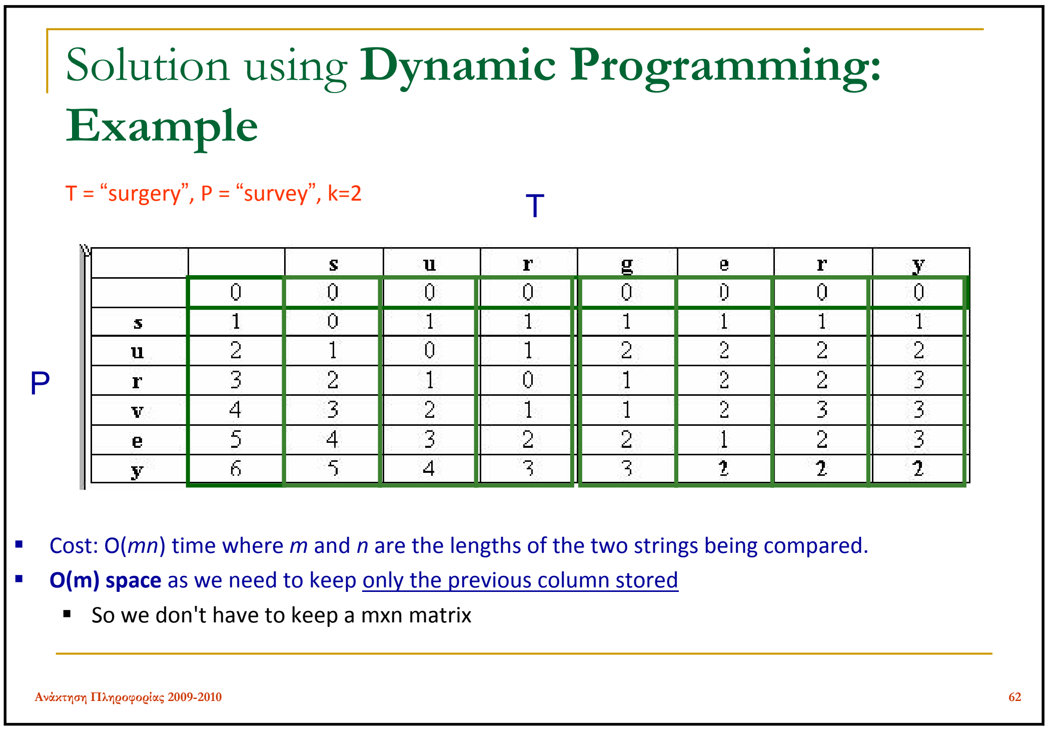
P

		s	u	r	g	e	r	y
	0	0	0	0	0	0	0	0
s	1	0	1	1	1	1	1	1
u	2	1	0	1	2	2	2	2
r	3	2	1	0	1	2	2	3
v	4	3	2	1	1	2	3	3
e	5	4	3	2	2	1	2	3
y	6	5	4	3	3	**2**	**2**	**2**

Bold entries indicate matching positions.

- Cost: O(*mn*) time where *m* and *n* are the lengths of the two strings being compared.
- Παρατήρηση: η πολυπλοκότητα είναι ανεξάρτητη του κ

Ανάκτηση Πληροφορίας 2009-2010 61



Solution using **Dynamic Programming: Example**

T = “surgery”, P = “survey”, k=2

T

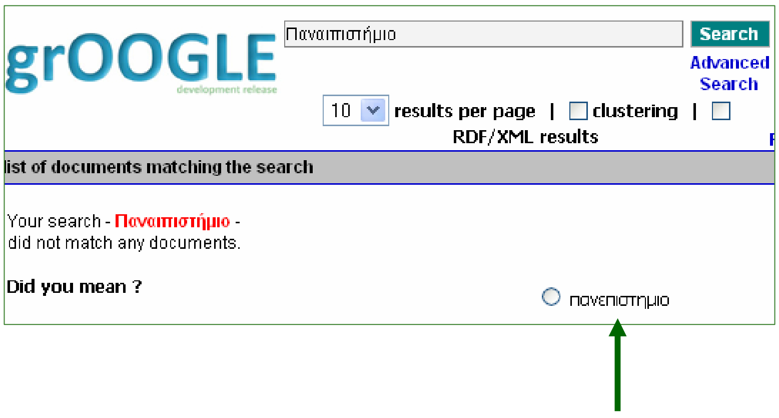
P

		s	u	r	g	e	r	y
	0	0	0	0	0	0	0	0
s	1	0	1	1	1	1	1	1
u	2	1	0	1	2	2	2	2
r	3	2	1	0	1	2	2	3
v	4	3	2	1	1	2	3	3
e	5	4	3	2	2	1	2	3
y	6	5	4	3	3	**2**	**2**	**2**

- Cost: O(*mn*) time where *m* and *n* are the lengths of the two strings being compared.
- **O(m) space** as we need to keep only the previous column stored
  - So we don't have to keep a mxn matrix

Ανάκτηση Πληροφορίας 2009-2010 62

Εφαρμογή στο groogle



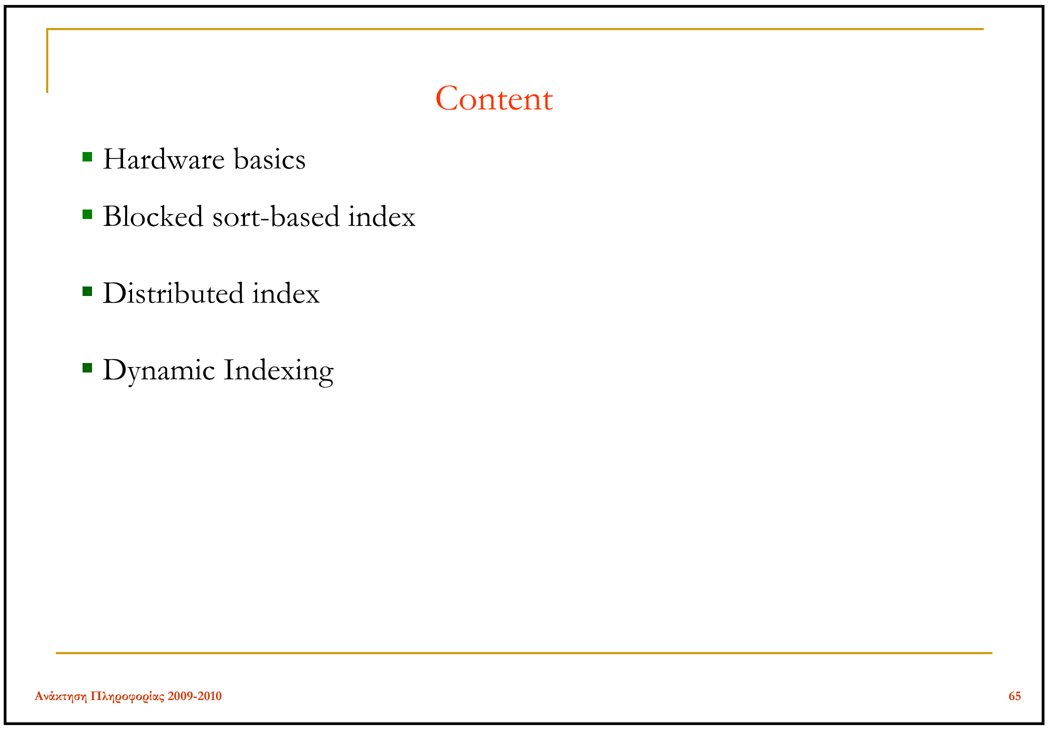
list of documents matching the search

Your search - Παναιπιστήμιο - did not match any documents.

Did you mean ?

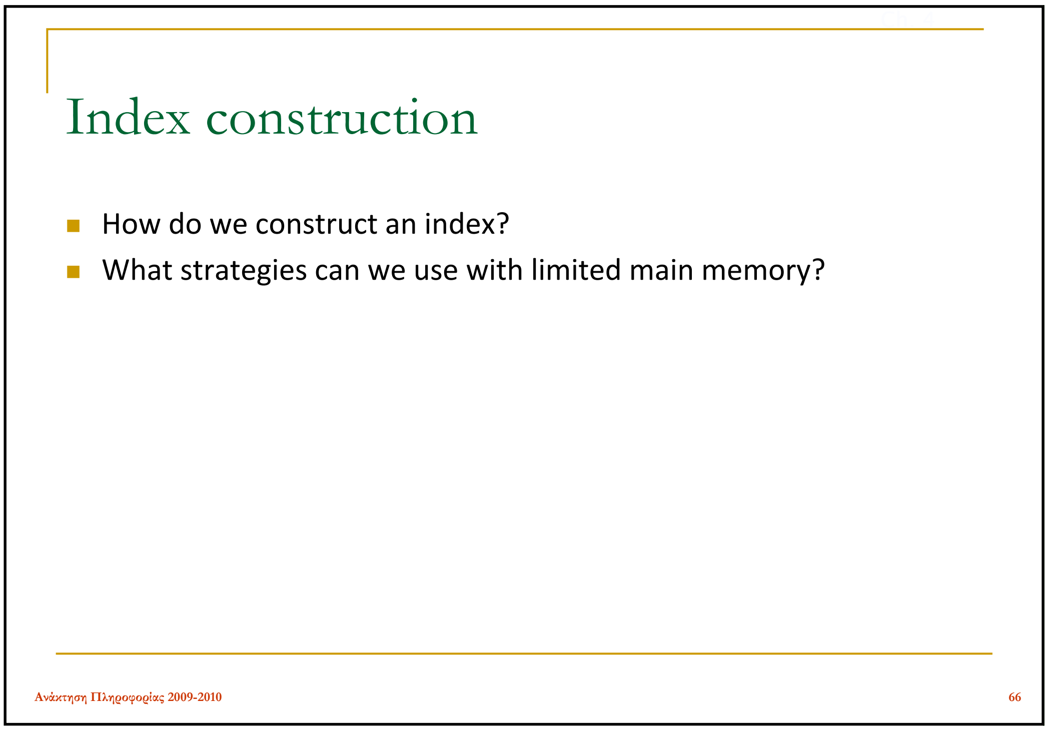
πανεπιστημιο

Constructing an Inverted File



Content

- Hardware basics
- Blocked sort-based index
- Distributed index
- Dynamic Indexing



Index construction

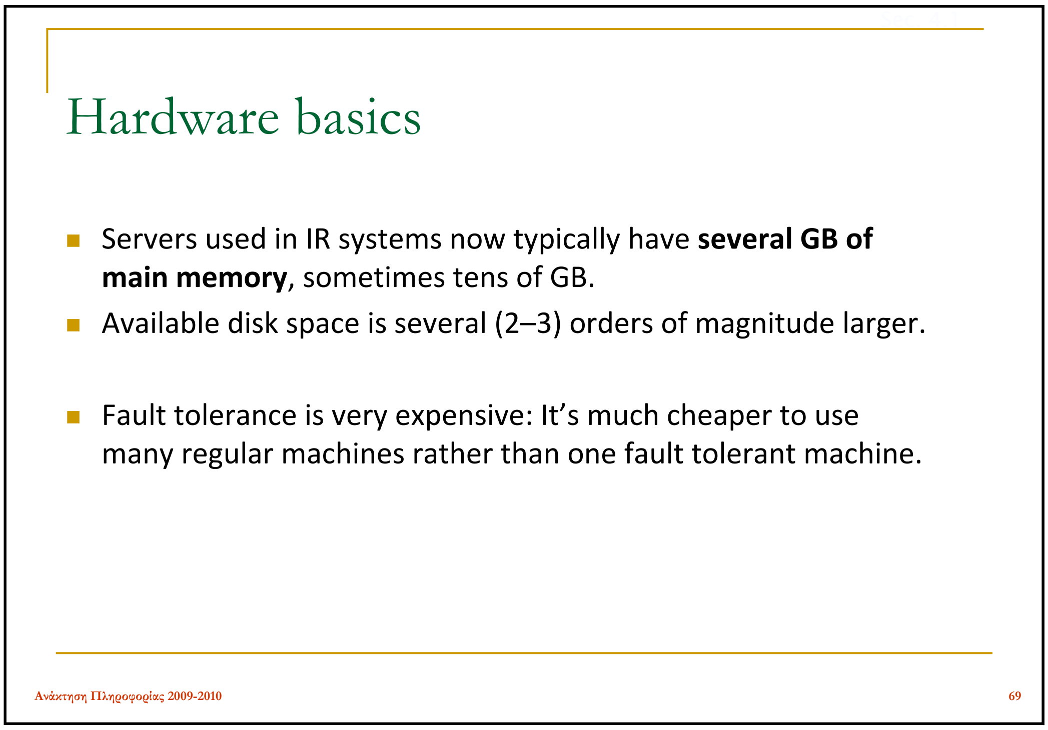
- How do we construct an index?
- What strategies can we use with limited main memory?

Hardware basics

- Many design decisions in information retrieval are based on the characteristics of hardware
- We begin by reviewing hardware basics

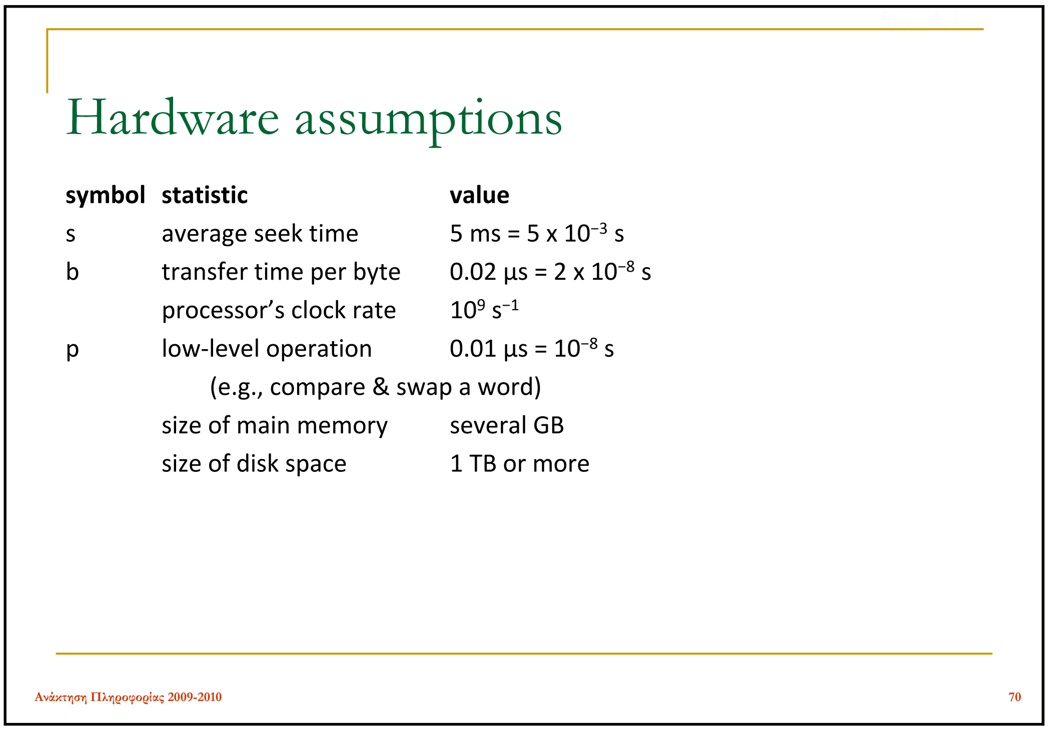
Hardware basics

- Access to data in memory is ***much*** faster than access to data on disk.
- *Caching (keeping frequently used disk data in memory)*
- Disk seeks: No data is transferred from disk while the disk head is being positioned. (seek time)
- *Therefore: Transferring one large chunk of data from disk to memory is faster than transferring many small chunks.*
- Disk I/O is **block-based**: Reading and writing of entire blocks (as opposed to smaller chunks).
- Block sizes: 8KB to 256 KB.
- Data transfers are handled by the system bus, not by the processor



Hardware basics

- Servers used in IR systems now typically have **several GB of main memory**, sometimes tens of GB.
- Available disk space is several (2–3) orders of magnitude larger.
- Fault tolerance is very expensive: It's much cheaper to use many regular machines rather than one fault tolerant machine.



Hardware assumptions

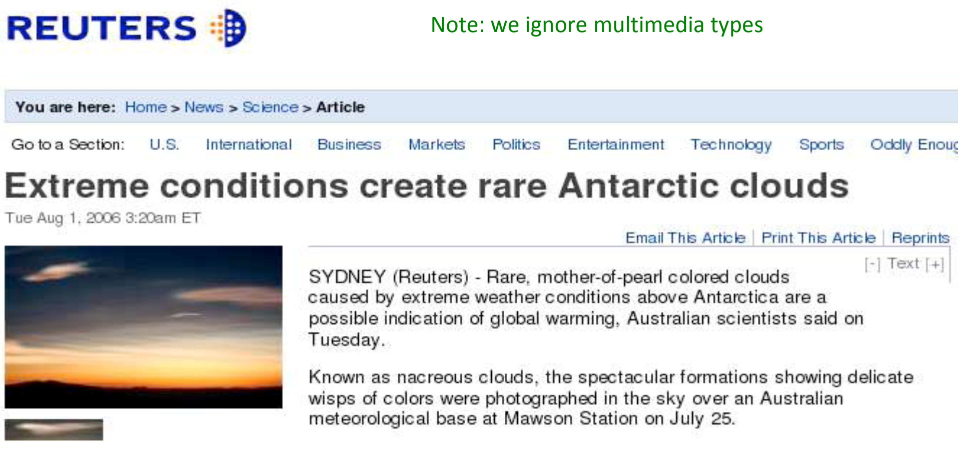
symbol	**statistic**	**value**
s	average seek time	5 ms = 5 x 10^{-3} s
b	transfer time per byte	0.02 μs = 2 x 10^{-8} s
	processor's clock rate	10^{9} s^{-1}
p	low-level operation (e.g., compare & swap a word)	0.01 μs = 10^{-8} s
	size of main memory	several GB
	size of disk space	1 TB or more

RCV1: Our collection for this lecture

- As an example for applying scalable index construction algorithms, we will use the Reuters RCV1 collection.
- This is one year of Reuters newswire (part of 1995 and 1996)
- The collection we'll use isn't really large enough either, but it's publicly available and is at least a more plausible example.

A Reuters RCV1 document

Note: we ignore multimedia types



REUTERS

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

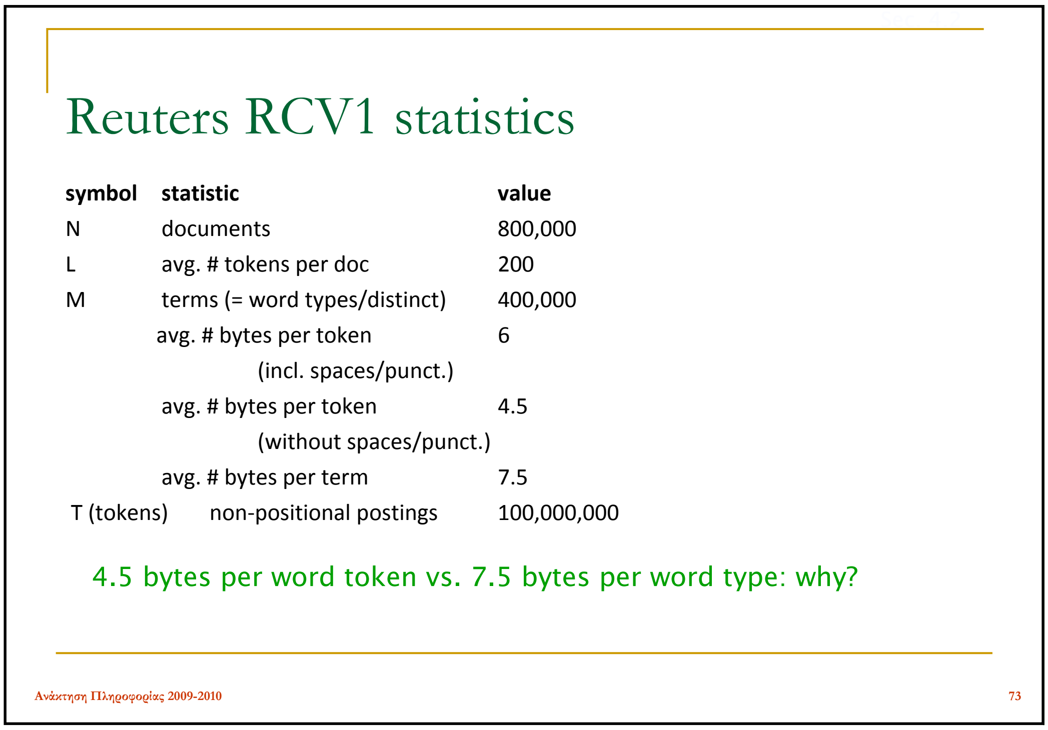
Tue Aug 1, 2006 3:20am ET

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.

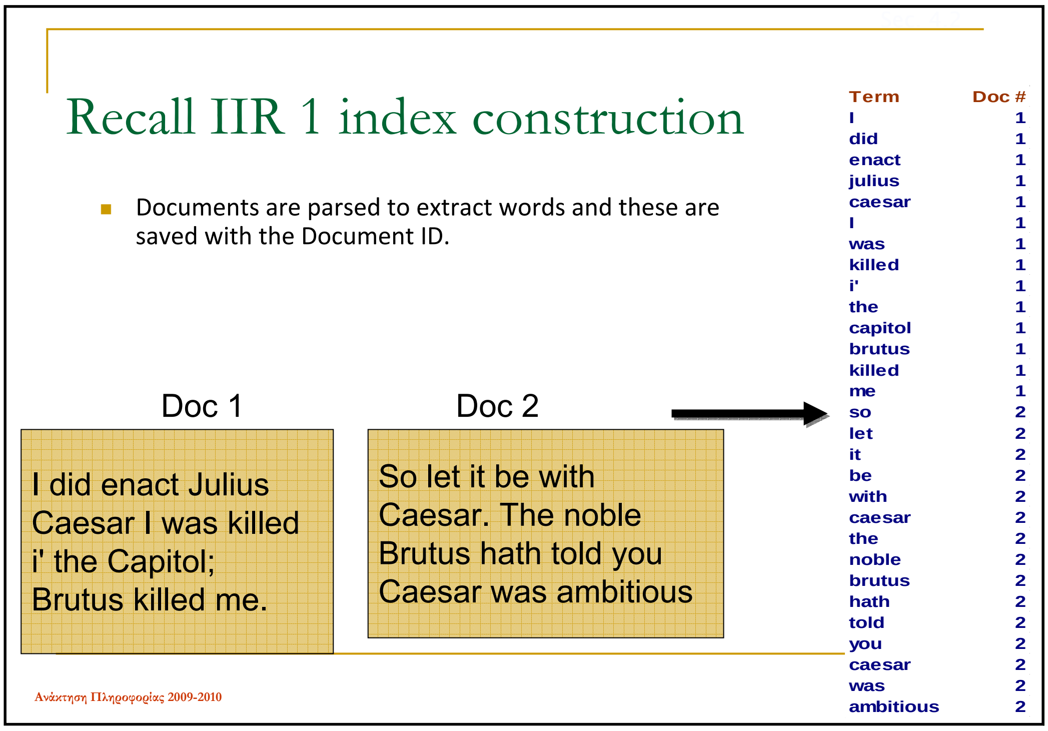


Reuters RCV1 statistics

symbol	statistic	value
N	documents	800,000
L	avg. # tokens per doc	200
M	terms (= word types/distinct)	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)	non-positional postings	100,000,000

4.5 bytes per word token vs. 7.5 bytes per word type: why?

Ανάκτηση Πληροφορίας 2009-2010 73



Recall IIR 1 index construction

- Documents are parsed to extract words and these are saved with the 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

Ανάκτηση Πληροφορίας 2009-2010

Key step

- After all documents have been parsed, the inverted file is **sorted by terms**.

We focus on this sort step.
We have 100M items to sort.

Note: present terms by termid

(on the fly or in two passes)

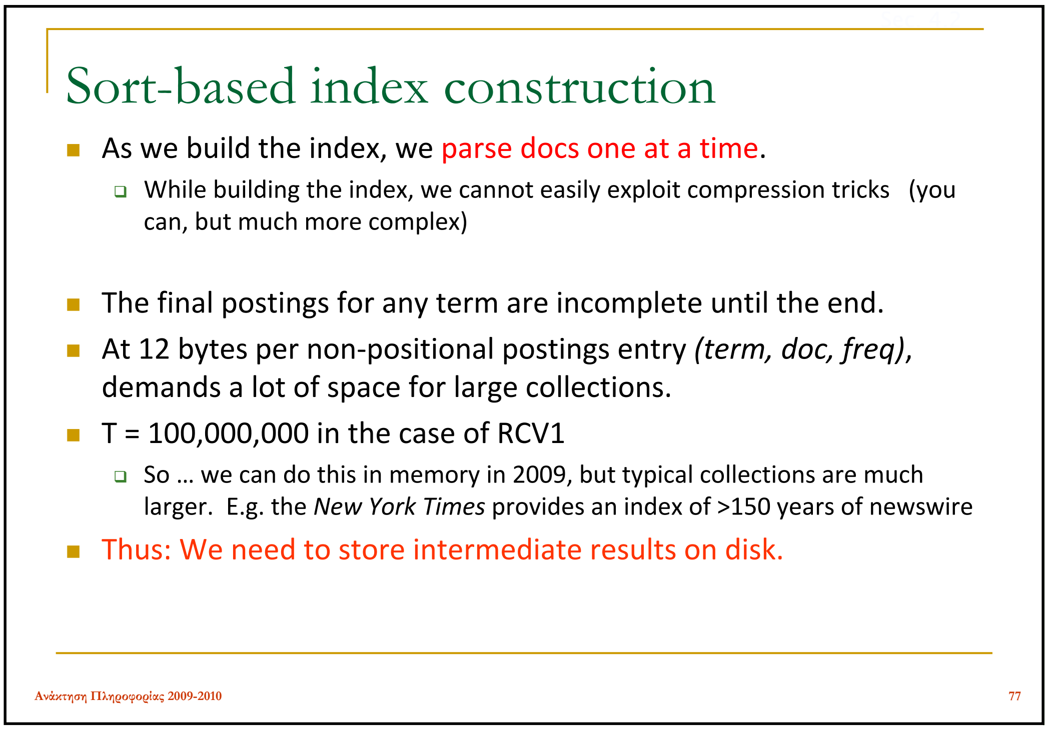
Term	Doc #
I	1
did	1
enact	1
julius	1
caesar	1
I	1
was	1
killed	1
i'	1
the	1
capitol	1
brutus	1
killed	1
me	1
so	2
let	2
it	2
be	2
with	2
caesar	2
the	2
noble	2
brutus	2
hath	2
told	2
you	2
caesar	2
was	2
ambitious	2

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

Scaling index construction

- In-memory index construction does not scale.
- How can we construct an index for very large collections?
- Taking into account the hardware constraints we just learned about . . .

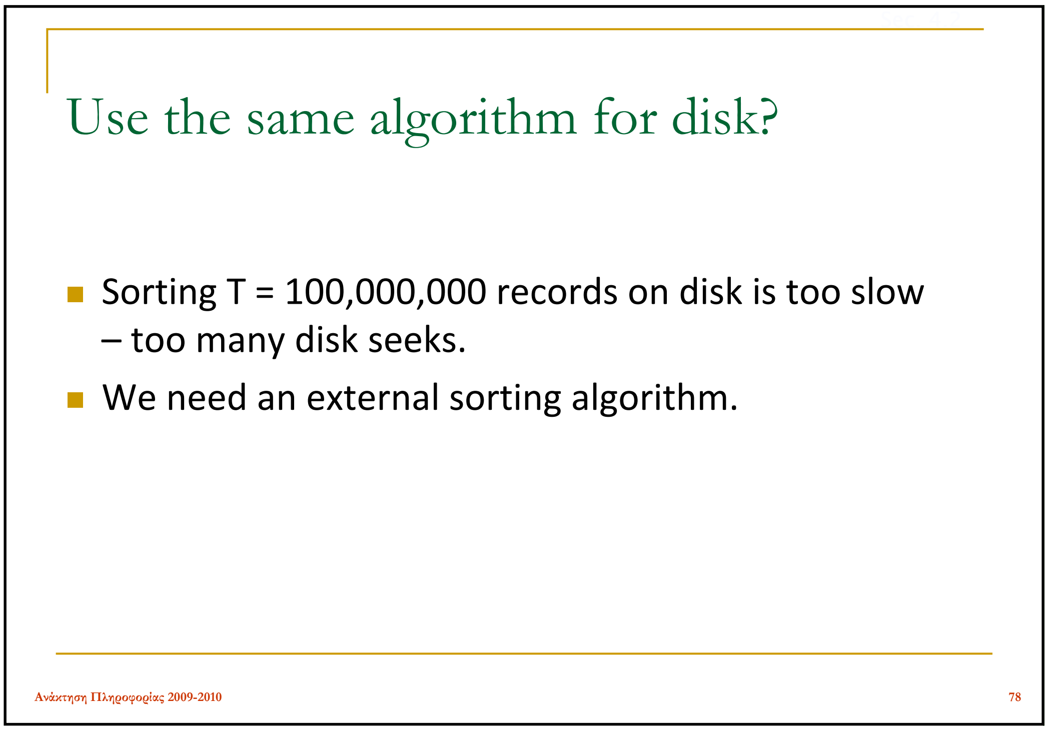
  Memory, disk, speed, etc.



Sort-based index construction

- As we build the index, we parse docs one at a time.
  - While building the index, we cannot easily exploit compression tricks (you can, but much more complex)
- The final postings for any term are incomplete until the end.
- At 12 bytes per non-positional postings entry *(term, doc, freq)*, demands a lot of space for large collections.
- T = 100,000,000 in the case of RCV1
  - So … we can do this in memory in 2009, but typical collections are much larger. E.g. the *New York Times* provides an index of >150 years of newswire
- Thus: We need to store intermediate results on disk.

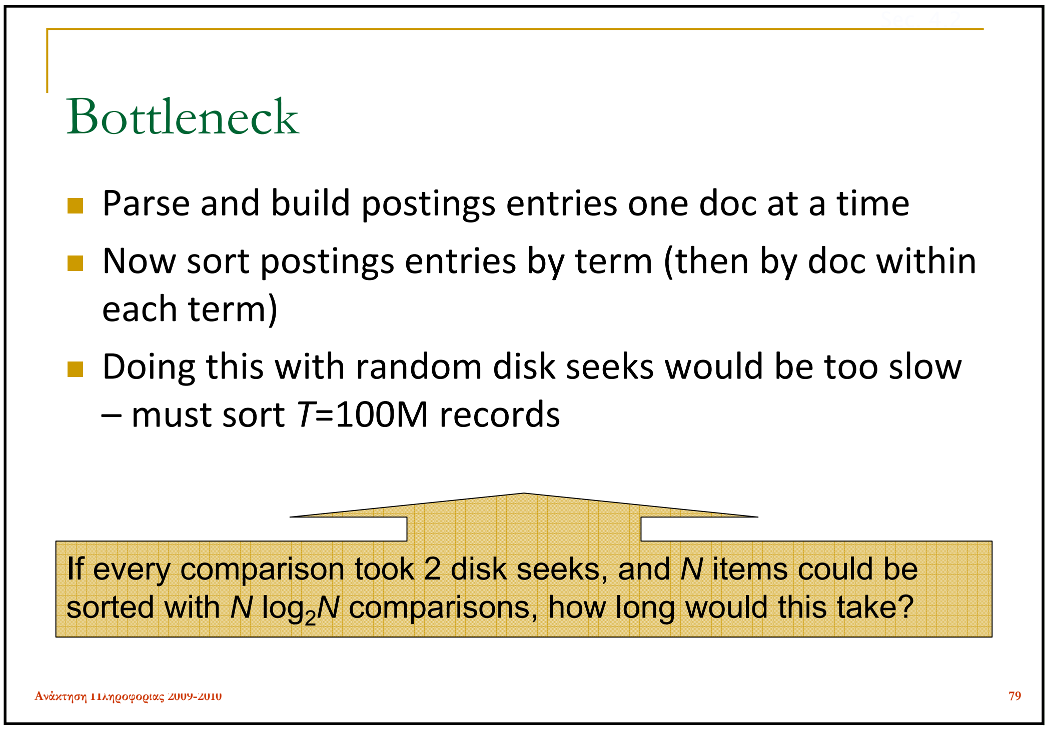
Ανάκτηση Πληροφορίας 2009-2010 77



Use the same algorithm for disk?

- Sorting T = 100,000,000 records on disk is too slow – too many disk seeks.
- We need an external sorting algorithm.

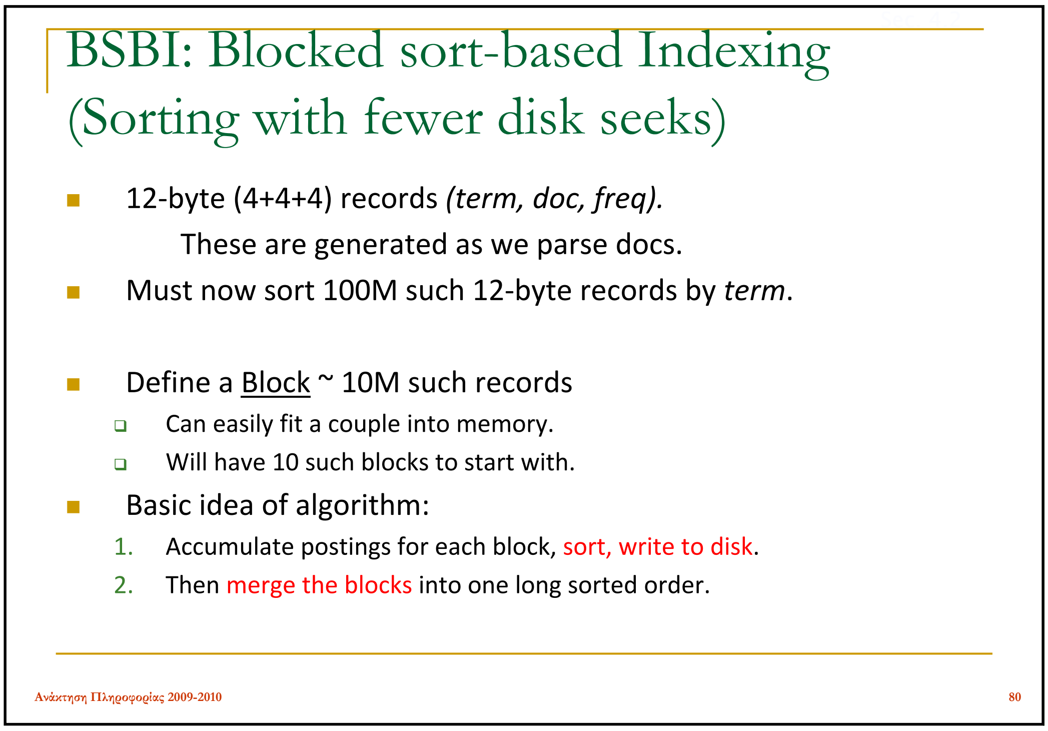
Ανάκτηση Πληροφορίας 2009-2010 78



Bottleneck

- Parse and build postings entries one doc at a time
- Now sort postings entries by term (then by doc within each term)
- Doing this with random disk seeks would be too slow – must sort *T*=100M records

If every comparison took 2 disk seeks, and *N* items could be sorted with $N \log_2 N$ comparisons, how long would this take?

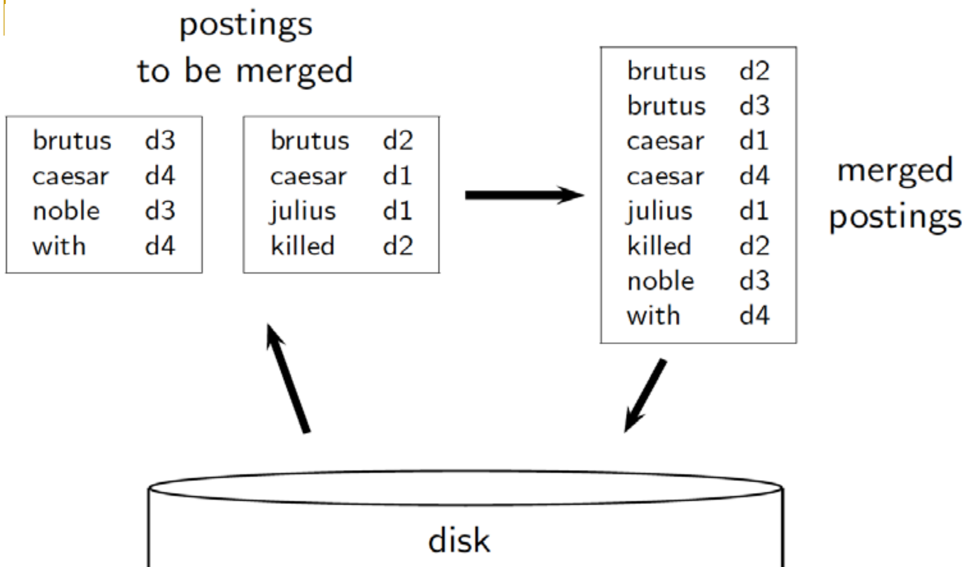


BSBI: Blocked sort-based Indexing (Sorting with fewer disk seeks)

- 12-byte (4+4+4) records *(term, doc, freq)*.

  These are generated as we parse docs.
- Must now sort 100M such 12-byte records by *term*.

- Define a Block ~ 10M such records
  - Can easily fit a couple into memory.
  - Will have 10 such blocks to start with.
- Basic idea of algorithm:
  1. Accumulate postings for each block, sort, write to disk.
  2. Then merge the blocks into one long sorted order.



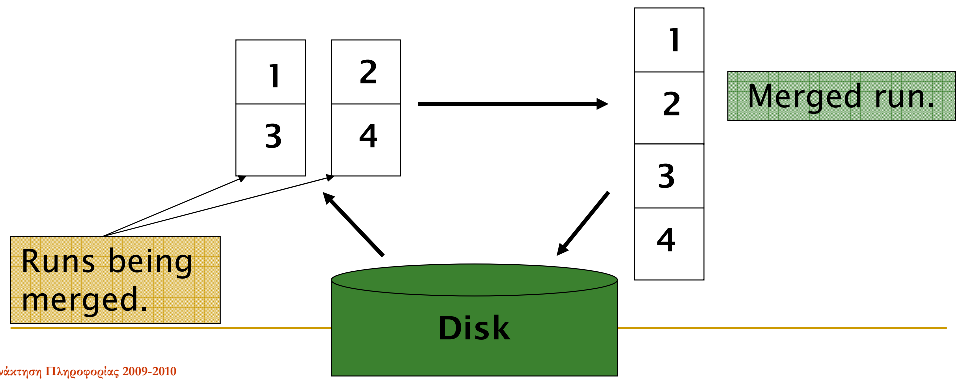
Sorting 10 blocks of 10M records

- First, read each block and sort within:
  - Quicksort takes 2*N* ln *N* expected steps
  - In our case 2 x (10M ln 10M) steps
- *total time to read each block from disk and quicksort it.*
- 10 times this estimate – gives us 10 sorted *runs* of 10M records each.
- Done straightforwardly, need 2 copies of data on disk
  - But can optimize this

```
BSBIndexConstruction()
1  $n \leftarrow 0$
2  while (all documents have not been processed)
3  do $n \leftarrow n + 1$
4      $block \leftarrow$ ParseNextBlock()
5      BSBI-Invert($block$)
6      WriteBlockToDisk($block, f_n$)
7  MergeBlocks($f_1, \ldots, f_n; f_{merged}$)
```

How to merge the sorted runs?

- Can do binary merges, with a merge tree of $\log_2 10 = 4$ layers.
- During each layer, read into memory runs in blocks of 10M, merge, write back.



How to merge the sorted runs?

- But it is more efficient to do a *n*-way merge, where you are *reading from all blocks simultaneously*
- Providing you read decent-sized chunks of each block into memory and then write out a decent-sized output chunk, then you're not killed by disk seeks

Remaining problem with sort-based algorithm

- Our assumption was: we can keep the dictionary in memory.
- *We need the dictionary (which grows dynamically) in order to implement a term to termID mapping.*

- Actually, we could work with term,docID postings instead of termID,docID postings . . .
- . . . but then intermediate files become very large. (We would end up with a scalable, but very slow index construction method.)

SPIMI: Single-pass in-memory indexing

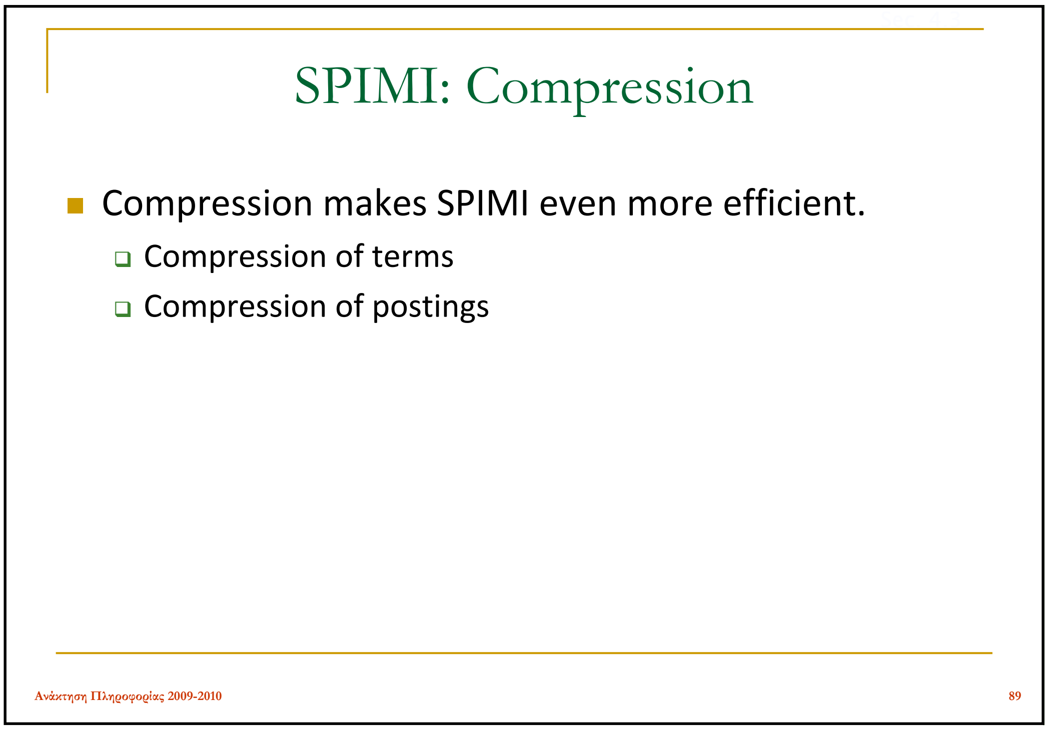
- Key idea 1: Generate **separate dictionaries for each block** – *no need to maintain term-termID mapping across blocks*.
- Key idea 2: Don't sort. Accumulate postings in postings lists as they occur.

- With these two ideas we can generate a complete inverted index for each block.
- These separate indexes can then be merged into one big index.

SPIMI-Invert

```
SPIMI-INVERT(token_stream)
output_file = NEWFILE()
dictionary = NEWHASH()
while (free memory available)
do token ← next(token_stream)
   if term(token) ∉ dictionary
     then postings_list = ADDTODICTIONARY(dictionary, term(token))
     else postings_list = GETPOSTINGSLIST(dictionary, term(token))
   if full(postings_list)
     then postings_list = DOUBLEPOSTINGSLIST(dictionary, term(token))
   ADDTOPOSTINGSLIST(postings_list, docID(token))
sorted_terms ← SORTTERMS(dictionary)
WRITEBLOCKTODISK(sorted_terms, dictionary, output_file)
return output_file
```

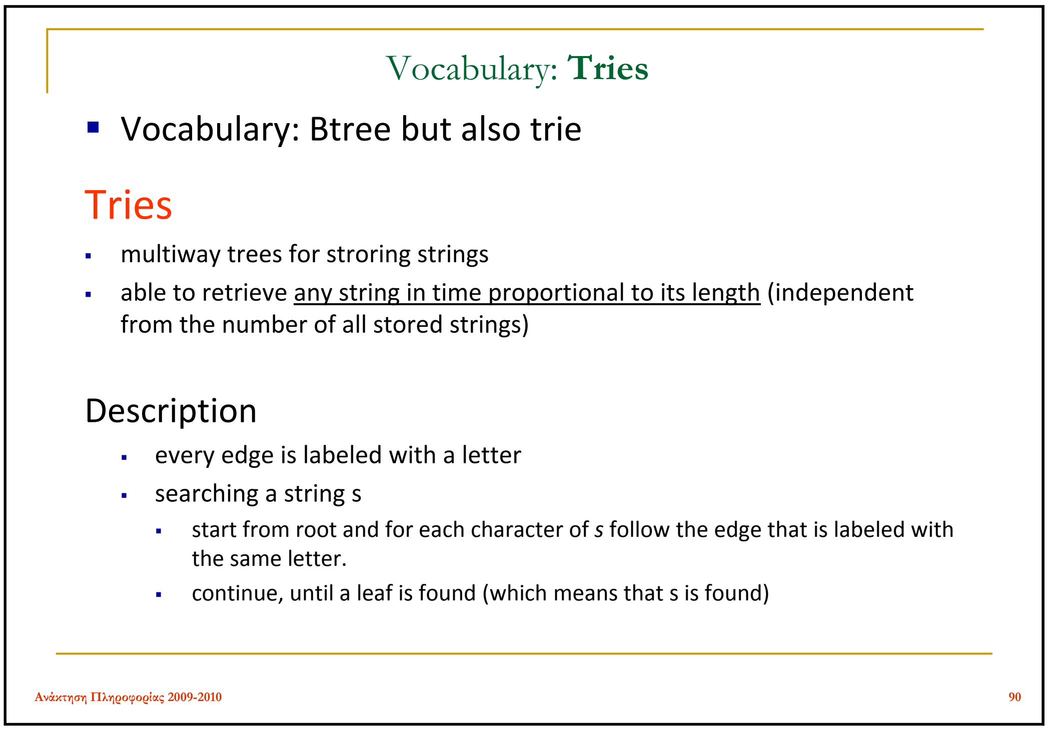
Merging of blocks is analogous to BSBI.



SPIMI: Compression

- Compression makes SPIMI even more efficient.
  - Compression of terms
  - Compression of postings

Ανάκτηση Πληροφορίας 2009-2010 89



Vocabulary: **Tries**

- Vocabulary: Btree but also trie

Tries

- multiway trees for stroring strings
- able to retrieve <u>any string in time proportional to its length</u> (independent from the number of all stored strings)

Description

- every edge is labeled with a letter
- searching a string s
  - start from root and for each character of *s* follow the edge that is labeled with the same letter.
  - continue, until a leaf is found (which means that s is found)

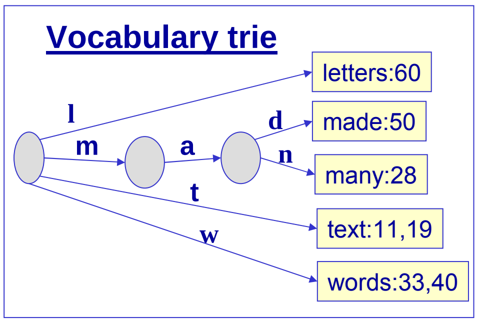
Ανάκτηση Πληροφορίας 2009-2010 90

Tries: Παράδειγμα

```
1    6   9 11  17 19  24   28    33      40      46  50   55    60
This is  a  text.  A text has  many words.  Words  are  made  from  letters.
```

Vocabulary
- text (11)
- text (19)
- many (28)
- words (33)
- words (40)
- made (50)
- letters (60)

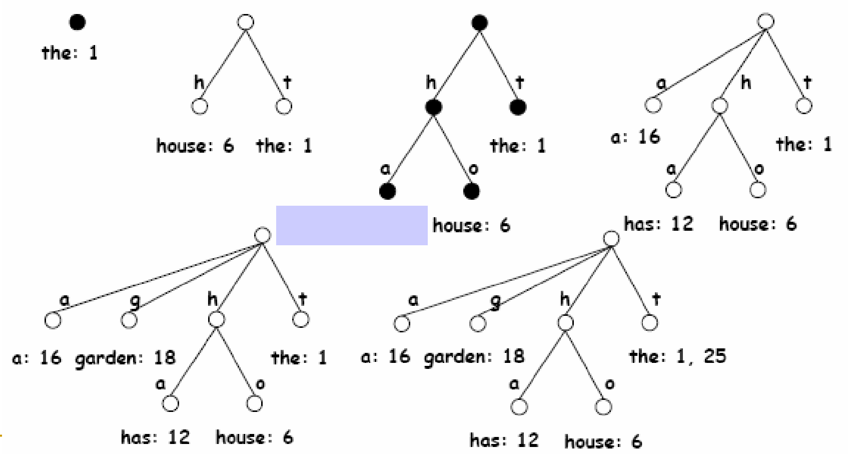
Vocabulary (ordered)
- letters (60)
- made (50)
- many (28)
- text (11,19)
- words (33,40)



Ερώτηση: Θα μπορούσε ένα trie να βοηθήσει τη στελέχωση κειμένου βάσει της τεχνικής Successor variety?

Παράδειγμα αυξητικής δημιουργίας ενός trie

```
1    6       12  16  18        25   29      36   40     45       54   58        66  70
the house has a  garden. the garden has many flowers. the flowers are beautiful
```
(each word = one document, position = document identifier)

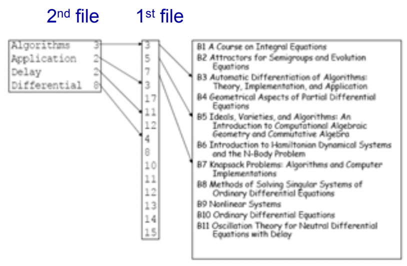


Constructing an Inverted File

- All the vocabulary is kept in a suitable data structure storing for each word a list of its occurrences
  - e.g. in a trie data structure
- Each word of the text is read and searched in the vocabulary
  - if a trie data structure is used then this search costs O(m) where m the size of the word
- If it is not found, it is **added** to the vocabulary with an empty list of occurrences and the **new position** is added to the end of its list of occurrences

Constructing an Inverted File

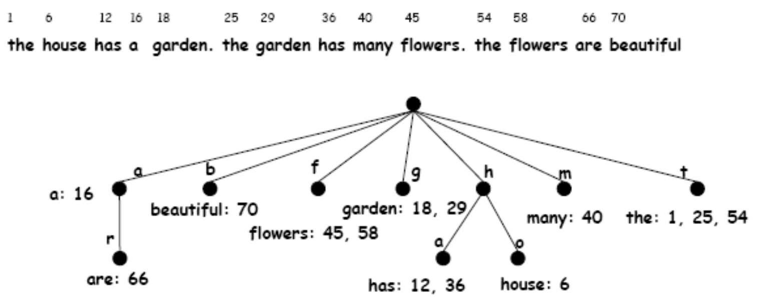
- Once the text is exhausted the vocabulary is written to disk with the list of occurrences. Two files are created:
  - in the <u>first file</u>, the list of occurrences are stored contiguously
  - in the <u>second file</u>, the vocabulary is stored in lexicographical order and, for each word, a pointer to its list in the first file is also included.
- The overall process is *O(n)* time


2nd file
1st file
Algorithms 3
Application 2
Delay 2
Differential 8
3
5
7
3
17
11
12
4
8
10
11
12
13
14
15
B1 A Course on Integral Equations
B2 Attractors for Semigroups and Evolution Equations
B3 Automatic Differentiation of Algorithms: Theory, Implementation, and Application
B4 Geometrical Aspects of Partial Differential Equations
B5 Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra
B6 Introduction to Hamiltonian Dynamical Systems and the N-Body Problem
B7 Knapsack Problems: Algorithms and Computer Implementations
B8 Methods of Solving Singular Systems of Ordinary Differential Equations
B9 Nonlinear Systems
B10 Ordinary Differential Equations
B11 Oscillation Theory for Neutral Differential Equations with Delay

Trie: O(1) per text character
Since positions are appended (in the postings file) O(1) time
It follows that the overall process is O(n)

Example of constructing an inverted file

(in our example we assume that: each word = one document, position = document identifier)



inverted file I

a: 16
are: 66
beautiful: 70
flowers: 45, 58
garden: 18, 29
has: 12, 36
house: 6
many: 40
the: 1, 25, 54

16, 66, 70, 45, 58, 18, 29, 12, 36, 6, 40

postings file

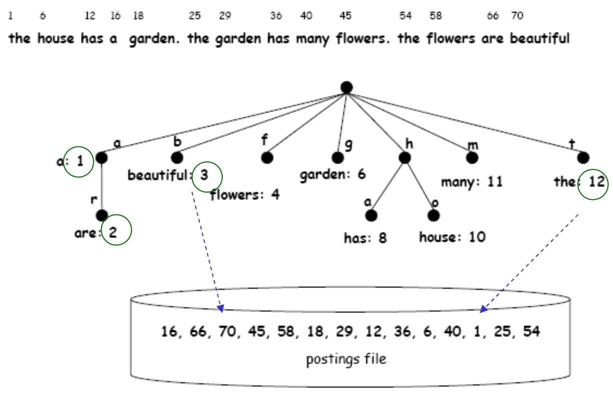
Once the complete trie structure is constructed the inverted file can be derived from it:

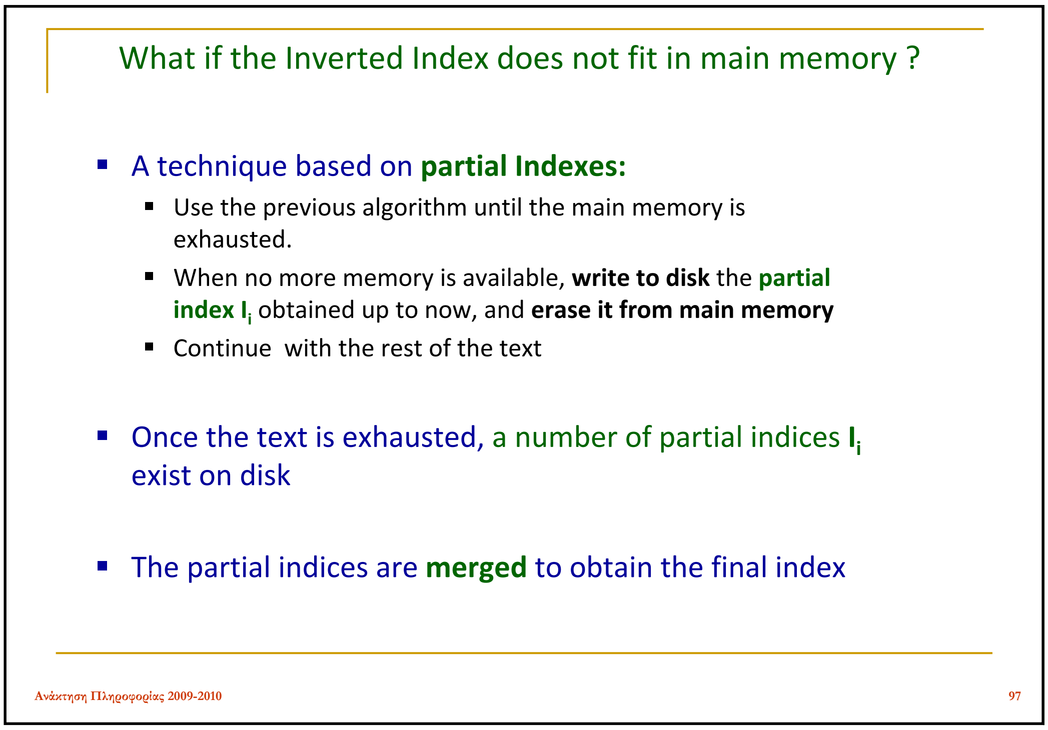
The trie is traversed top-down and left-to-right.

- whenever an index term is encountered, it is **added to the end of the inverted file**. Note that if a term is prefix of another term (such as "a" is prefix of "are") index terms can occur on internal nodes of the trie.
- analogously the posting file is derived.

Example (cont)

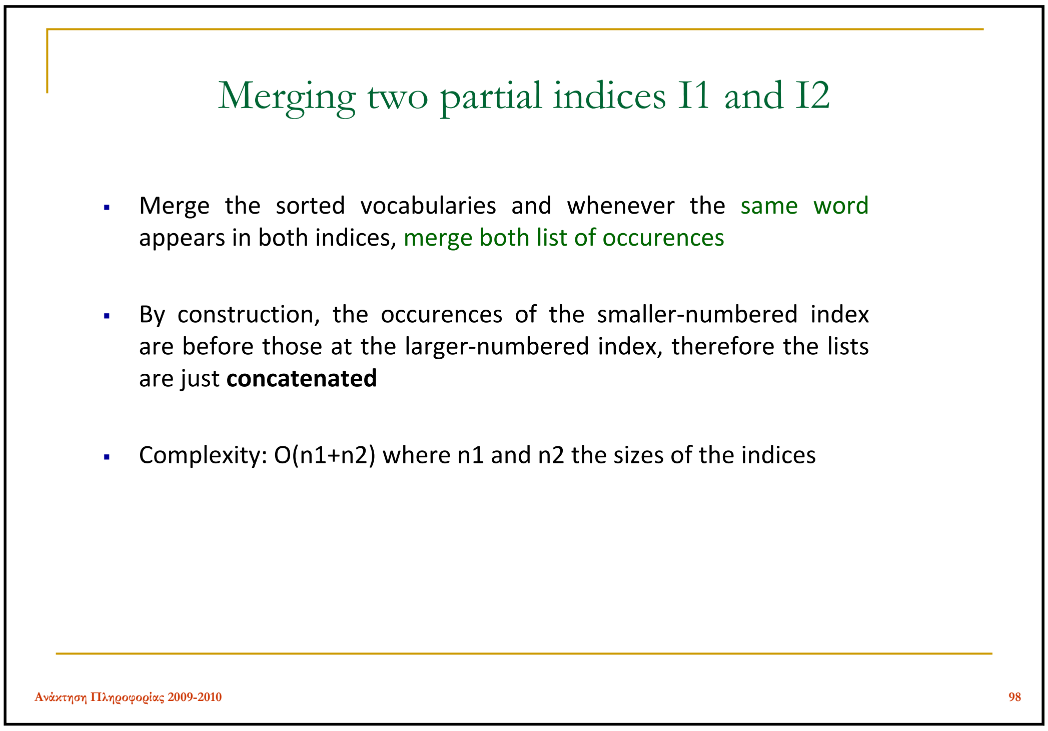
The trie structure constructed is a possible access structure to the index file in main memory. Thus the entries of the index files occur as leaves (or internal nodes) of the trie. Each entry has a reference to the position of the postings file that is held in secondary storage.





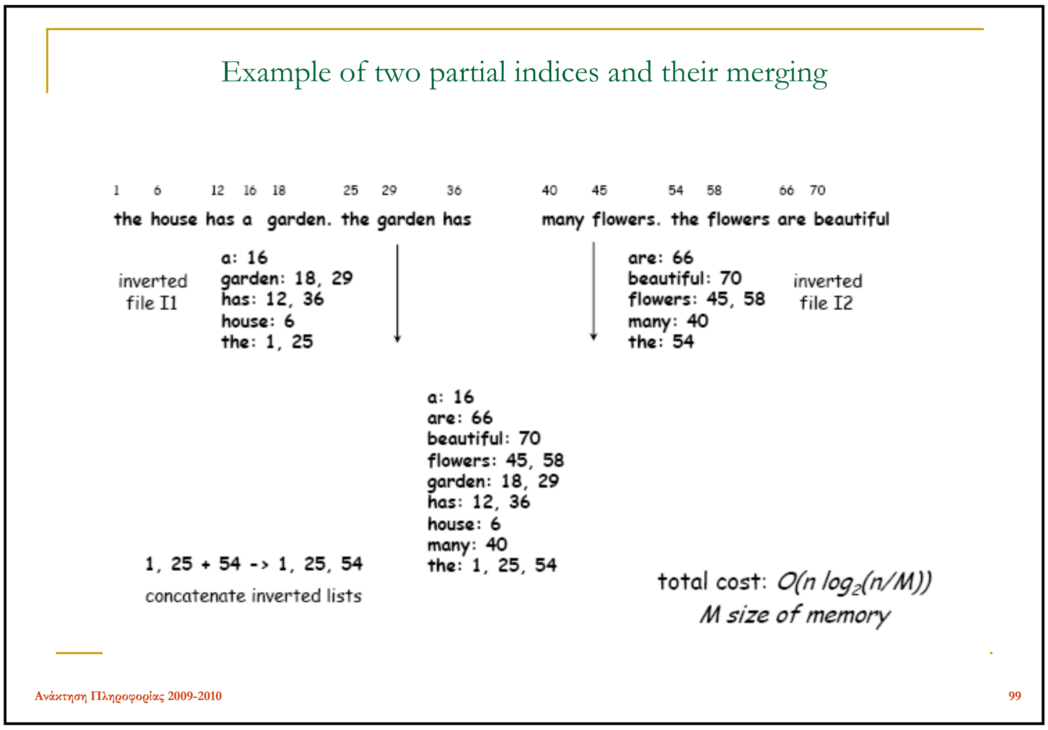
What if the Inverted Index does not fit in main memory ?

- A technique based on **partial Indexes:**
  - Use the previous algorithm until the main memory is exhausted.
  - When no more memory is available, **write to disk** the **partial index I_i** obtained up to now, and **erase it from main memory**
  - Continue with the rest of the text
- Once the text is exhausted, a number of partial indices **I_i** exist on disk
- The partial indices are **merged** to obtain the final index



Merging two partial indices I1 and I2

- Merge the sorted vocabularies and whenever the same word appears in both indices, merge both list of occurences
- By construction, the occurences of the smaller-numbered index are before those at the larger-numbered index, therefore the lists are just **concatenated**
- Complexity: O(n1+n2) where n1 and n2 the sizes of the indices



Example of two partial indices and their merging

```
1    6      12  16  18       25   29       36
the house has a  garden. the garden has
```

```
40    45        54    58       66   70
many flowers. the flowers are beautiful
```

inverted file I1

a: 16
garden: 18, 29
has: 12, 36
house: 6
the: 1, 25

inverted file I2

are: 66
beautiful: 70
flowers: 45, 58
many: 40
the: 54

a: 16
are: 66
beautiful: 70
flowers: 45, 58
garden: 18, 29
has: 12, 36
house: 6
many: 40
the: 1, 25, 54

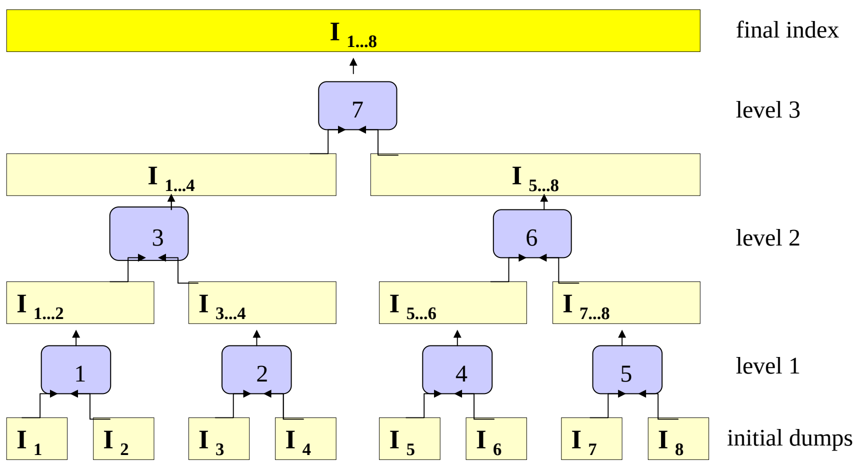
1, 25 + 54 -> 1, 25, 54

concatenate inverted lists

total cost: *$O(n \log_2(n/M))$*

M size of memory

Merging partial indices to obtain the final



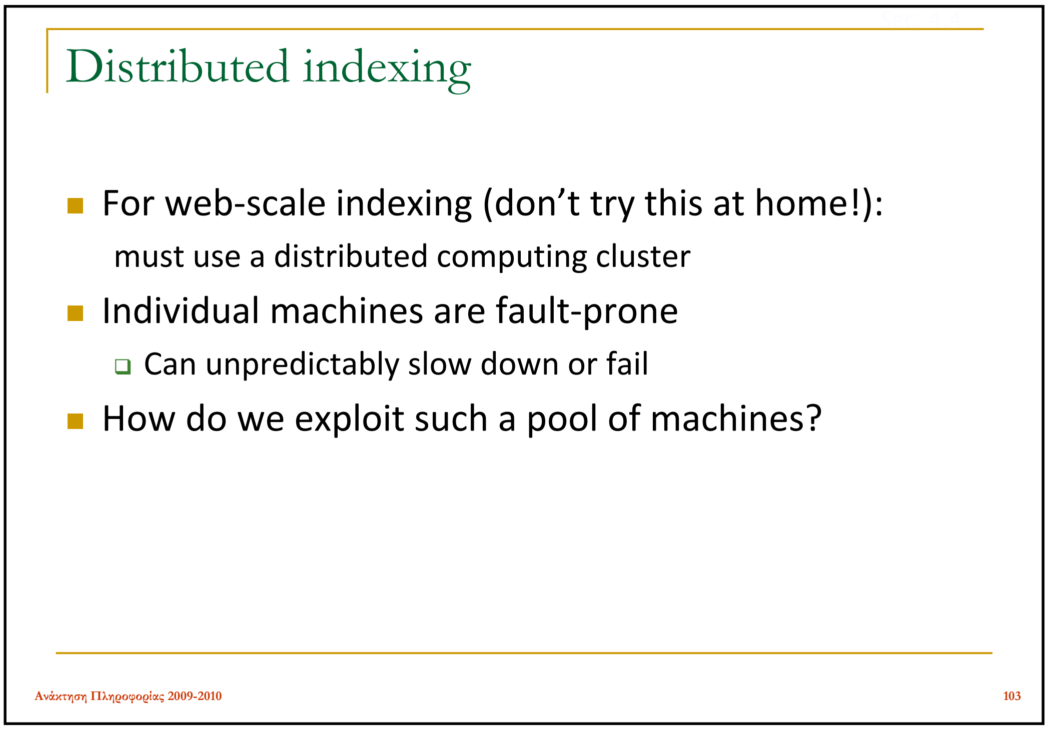
Merging all partial indices: Time Complexity

Notations

- *n*: the size of the text
- V: the size of the vocabulary
- *M*: the amount of main memory available

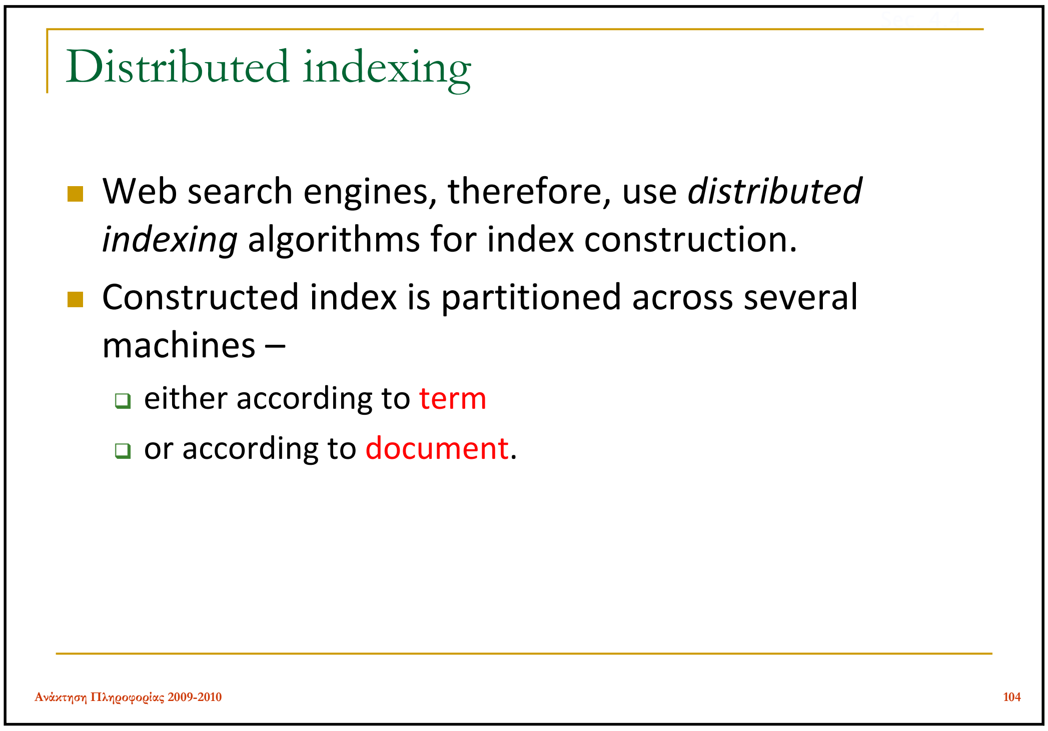
- The total time to generate partial indices is $O(n)$
- The number of partial indices is $O(n/M)$
- To merge the $O(n/M)$ partial indices are necessary $log_2(n/M)$ merging levels
- The total cost of this algorithm is $O(n \log(n/M))$

Distributed indexing



Distributed indexing

- For web-scale indexing (don’t try this at home!):
  must use a distributed computing cluster
- Individual machines are fault-prone
  - Can unpredictably slow down or fail
- How do we exploit such a pool of machines?



Distributed indexing

- Web search engines, therefore, use *distributed indexing* algorithms for index construction.
- Constructed index is partitioned across several machines –
  - either according to term
  - or according to document.

Google data centers

- Google data centers mainly contain commodity machines.
- Data centers are distributed around the world.
- Estimate: a total of 1 million servers, 3 million processors/cores (Gartner 2007)
- Estimate: Google installs 100,000 servers each quarter.
  - Based on expenditures of 200–250 million dollars per year
- This would be 10% of the computing capacity of the world!?!

Google data centers

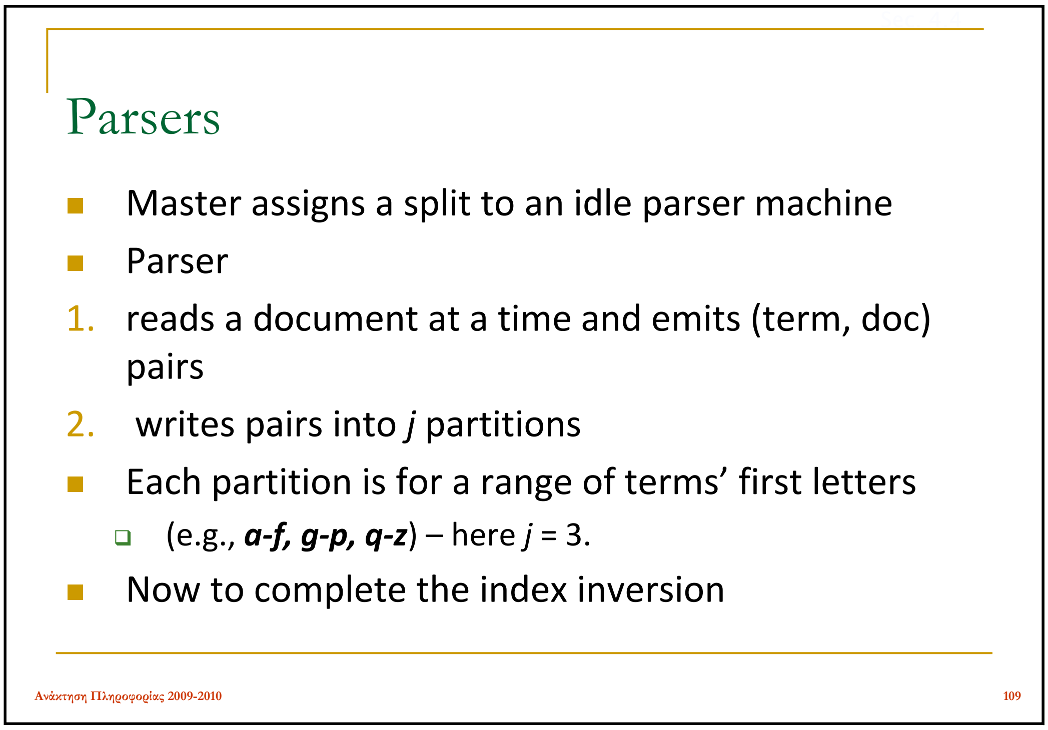
- If in a non-fault-tolerant system with 1000 nodes, each node has 99.9% uptime, what is the uptime of the system?
- Answer: 63%
- Calculate the number of servers failing per minute for an installation of 1 million servers.

Distributed indexing

- 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
- Divide data into chunks, re-assign task if a task on a chunk fails

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)



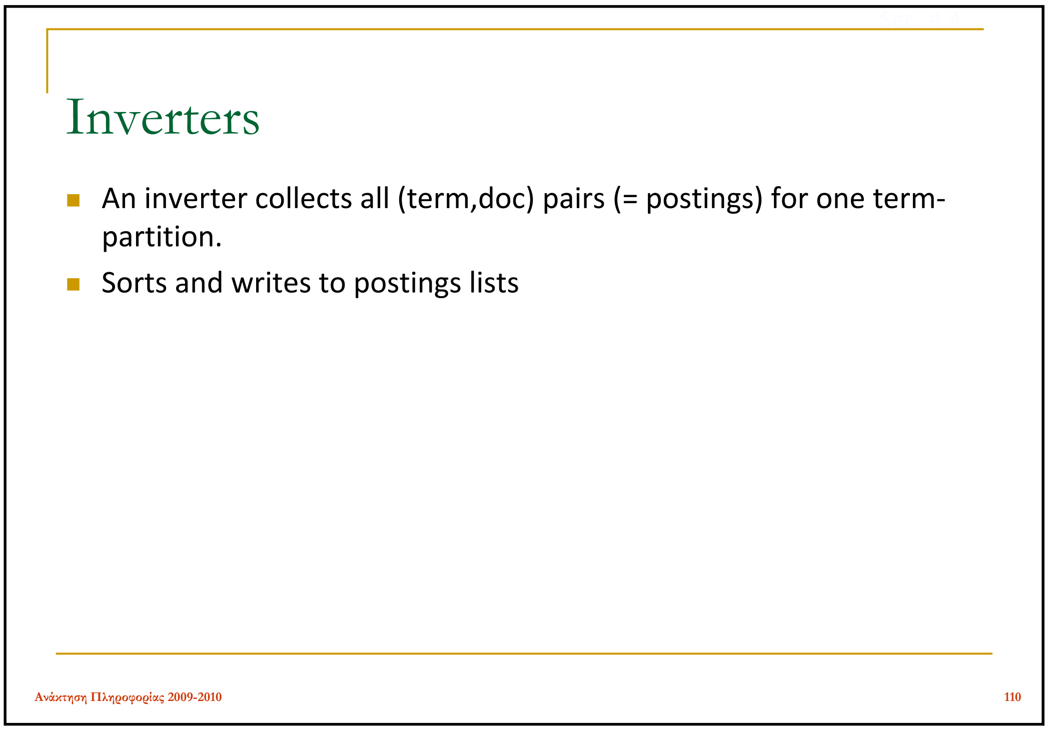
Parsers

- Master assigns a split to an idle parser machine
- Parser

1. reads a document at a time and emits (term, doc) pairs
2. 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

Ανάκτηση Πληροφορίας 2009-2010 109

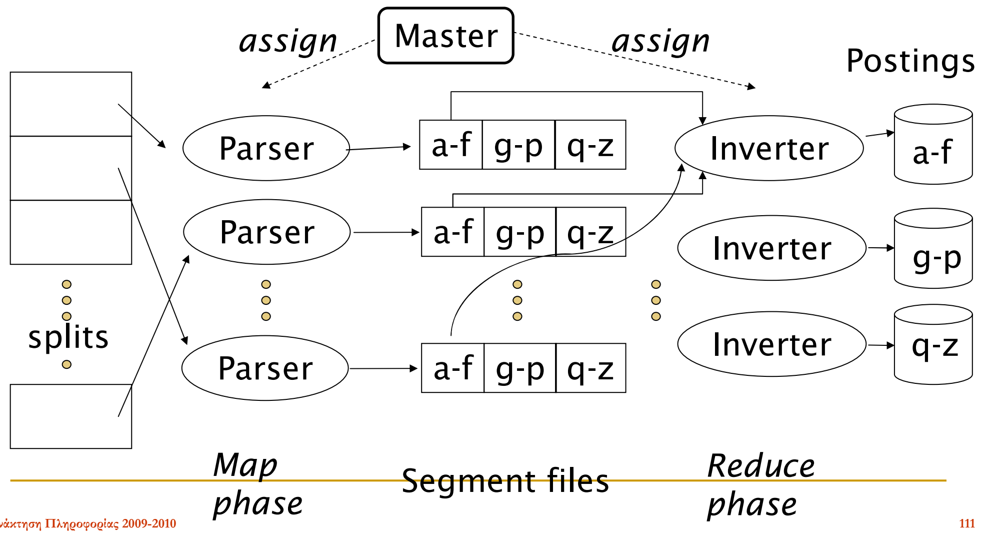


Inverters

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

Ανάκτηση Πληροφορίας 2009-2010 110

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.

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.

MapReduce

- To minimize write times, before inverters reduce the data, each parser writes its segment files to its *local disk*.
- In the reduce phase, the master communicates to an inverter **the locations of the relevant segment files** (e.g., of the *r* segment files of the a–f partition).
- Each segment file only requires one sequential read because all data relevant to a particular inverter were written to a single segment file by the parser. This setup minimizes the amount of network traffic needed during indexing.

Schema for index construction in MapReduce

(Key, Value) pairs

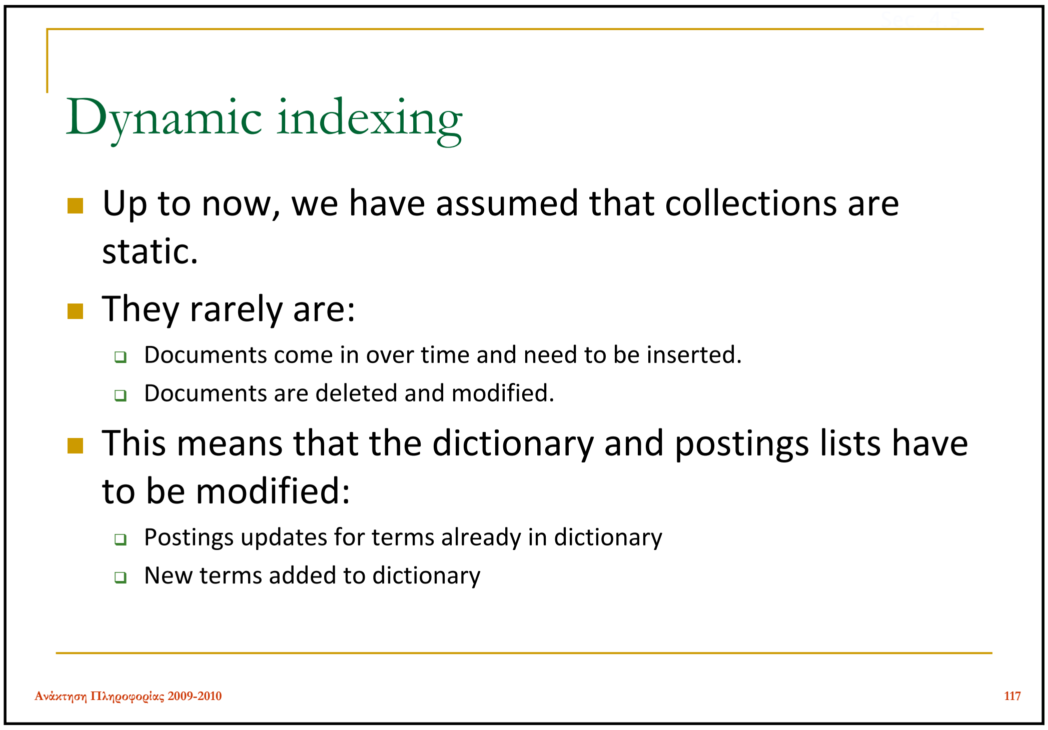
- **Schema of map and reduce functions**

  map: input → list(k, v)   reduce: (k,list(v)) → output
- **Instantiation of the schema for index construction**

  map: web collection → list(termID, docID)

  reduce: (<termID1, list(docID)>, <termID2, list(docID)>, …) → (postings list1, postings list2, …)
- **Example for index construction**
- map: d2 : C died. d1 : C came, C c'ed. → (<C, d2>, <died,d2>, <C,d1>, <came,d1>, <C,d1>, <c'ed, d1>
- reduce: (<C,(d2,d1,d1)>, <died,(d2)>, <came,(d1)>, <c'ed,(d1)>) → (<C,(d1:2,d2:1)>, <died,(d2:1)>, <came,(d1:1)>, <c'ed,(d1:1)>)

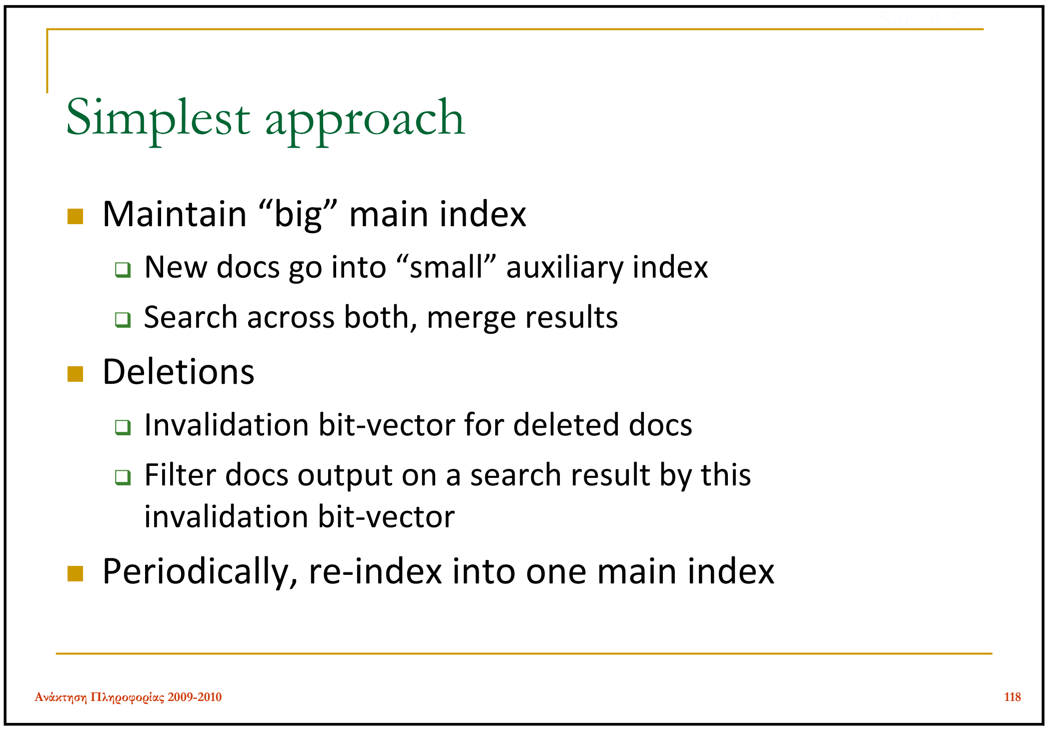
Dynamic indexing



Dynamic indexing

- Up to now, we have assumed that collections are static.
- They rarely are:
  - Documents come in over time and need to be inserted.
  - Documents are deleted and modified.
- This means that the dictionary and postings lists have to be modified:
  - Postings updates for terms already in dictionary
  - New terms added to dictionary

Ανάκτηση Πληροφορίας 2009-2010 117



Simplest approach

- Maintain "big" main index
  - New docs go into "small" auxiliary index
  - Search across both, merge results
- Deletions
  - Invalidation bit-vector for deleted docs
  - Filter docs output on a search result by this invalidation bit-vector
- Periodically, re-index into one main index

Ανάκτηση Πληροφορίας 2009-2010 118

Issues with main and auxiliary indexes

- Problem of frequent merges – you touch stuff a lot
- Poor performance during merge
- Actually:
  - Merging of the auxiliary index into the main index is efficient if we keep a separate file for each postings list.
  - Merge is the same as a simple append.
  - But then we would need a lot of files – inefficient for O/S.
- Assumption for the rest of the lecture: The index is one big file.
- In reality: Use a scheme somewhere in between (e.g., split very large postings lists, collect postings lists of length 1 in one file etc.)

Logarithmic merge

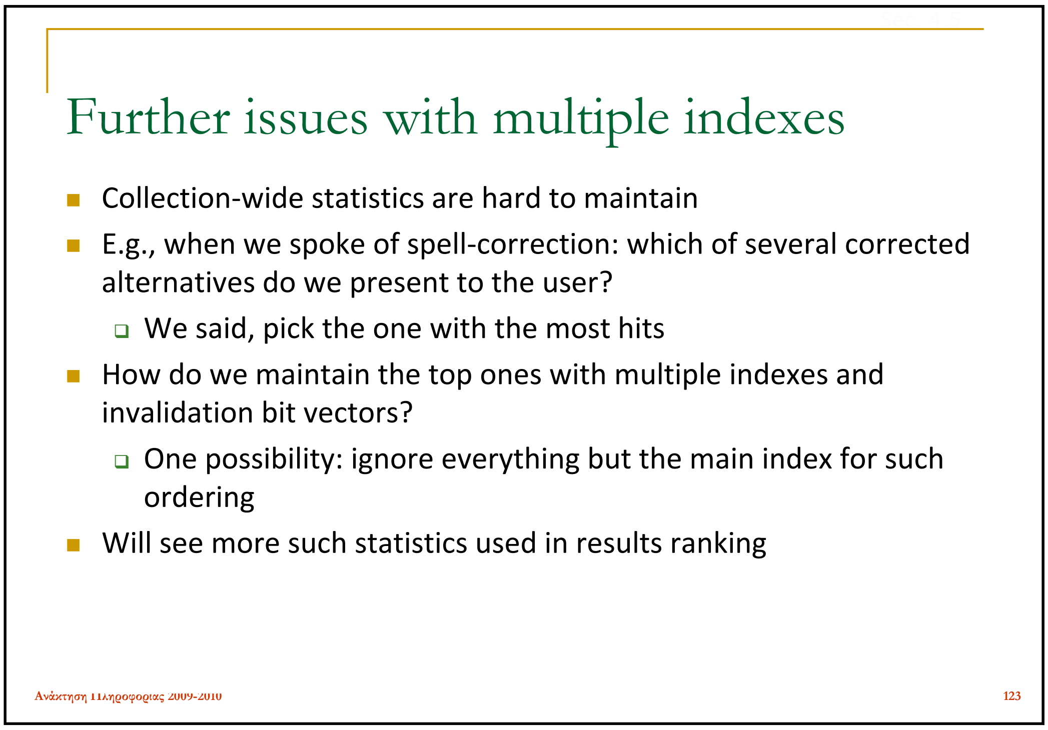
- Maintain a series of indexes, each twice as large as the previous one.
- Keep smallest (Z_0) in memory
- Larger ones (I_0, I_1, …) on disk
- If Z_0 gets too big (> *n*), write to disk as I_0
- or merge with I_0 (if I_0 already exists) as Z_1
- Either write merge Z_1 to disk as I_1 (if no I_1)
- Or merge with I_1 to form Z_2
- etc.

```
LMergeAddToken($indexes, Z_0, token$)
 1  $Z_0 \leftarrow$ Merge($Z_0, \{token\}$)
 2  if $|Z_0| = n$
 3    then for $i \leftarrow 0$ to $\infty$
 4           do if $I_i \in indexes$
 5                then $Z_{i+1} \leftarrow$ Merge($I_i, Z_i$)
 6                       ($Z_{i+1}$ is a temporary index on disk.)
 7                     $indexes \leftarrow indexes - \{I_i\}$
 8                else $I_i \leftarrow Z_i$    ($Z_i$ becomes the permanent index $I_i$.)
 9                     $indexes \leftarrow indexes \cup \{I_i\}$
10                     Break
11         $Z_0 \leftarrow \emptyset$

LogarithmicMerge()
1  $Z_0 \leftarrow \emptyset$    ($Z_0$ is the in-memory index.)
2  $indexes \leftarrow \emptyset$
3  while true
4  do LMergeAddToken($indexes, Z_0$, getNextToken())
```

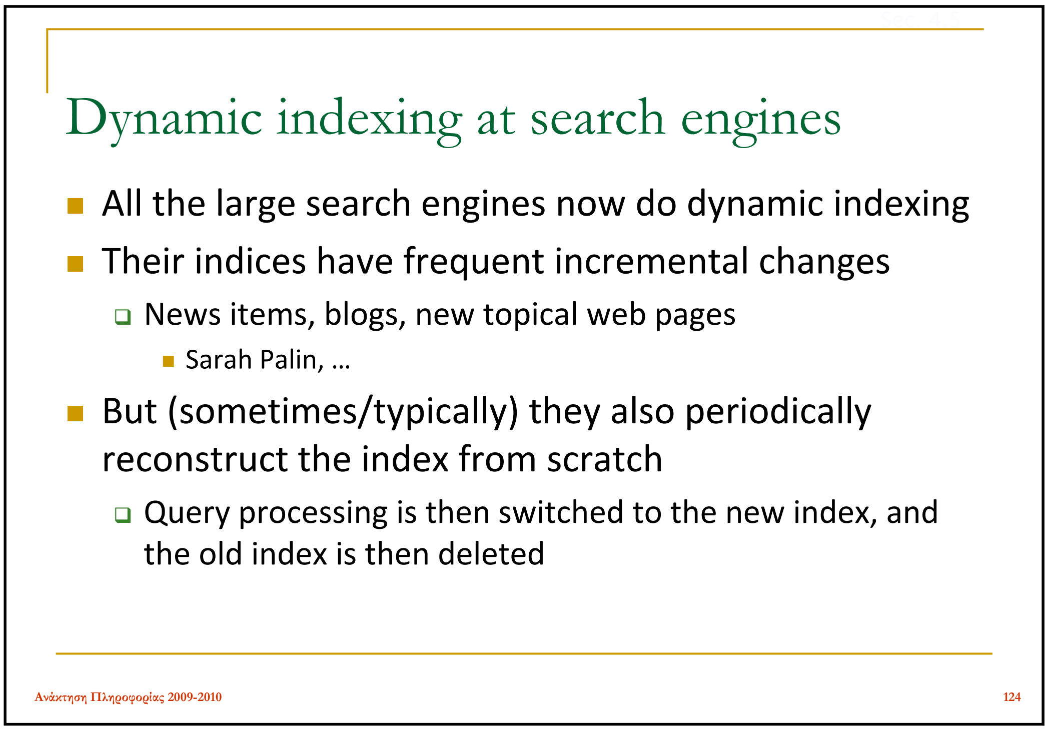
Logarithmic merge

- Auxiliary and main index: index construction time is O(T^2) as each posting is touched in each merge.
- Logarithmic merge: Each posting is merged O(log T) times, so complexity is O(T log T)
- So logarithmic merge is much more efficient for index construction
- But query processing now requires the merging of O(log T) indexes
  - Whereas it is O(1) if you just have a main and auxiliary index



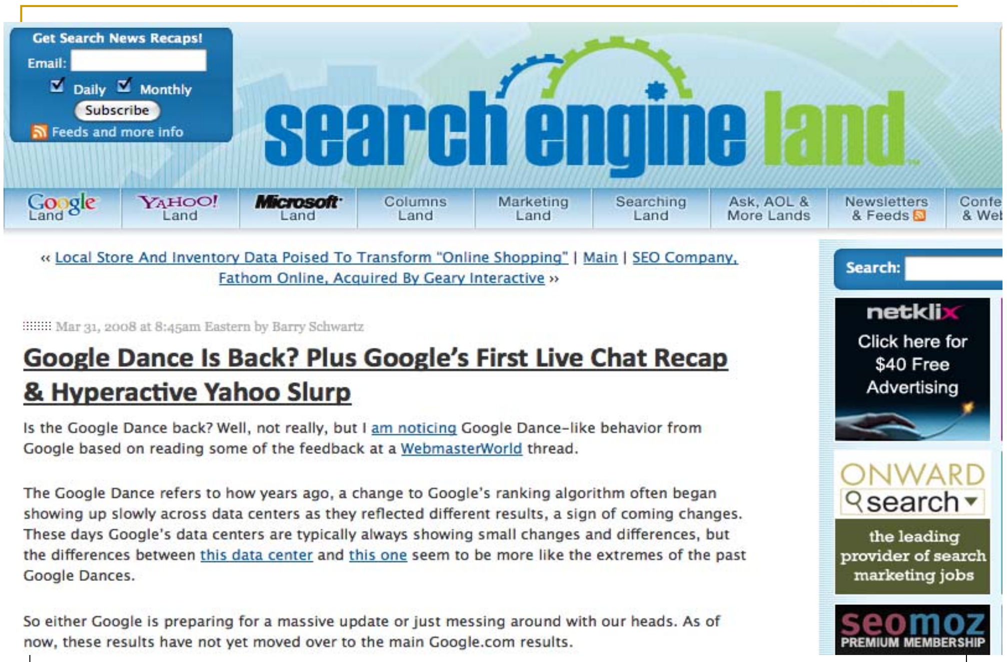
Further issues with multiple indexes

- Collection-wide statistics are hard to maintain
- E.g., when we spoke of spell-correction: which of several corrected alternatives do we present to the user?
  - We said, pick the one with the most hits
- How do we maintain the top ones with multiple indexes and invalidation bit vectors?
  - One possibility: ignore everything but the main index for such ordering
- Will see more such statistics used in results ranking



Dynamic indexing at search engines

- All the large search engines now do dynamic indexing
- Their indices have frequent incremental changes
  - News items, blogs, new topical web pages
    - Sarah Palin, …
- But (sometimes/typically) they also periodically reconstruct the index from scratch
  - Query processing is then switched to the new index, and the old index is then deleted



« Local Store And Inventory Data Poised To Transform "Online Shopping" | Main | SEO Company, Fathom Online, Acquired By Geary Interactive »

Mar 31, 2008 at 8:45am Eastern by Barry Schwartz

Google Dance Is Back? Plus Google's First Live Chat Recap & Hyperactive Yahoo Slurp

Is the Google Dance back? Well, not really, but I am noticing Google Dance-like behavior from Google based on reading some of the feedback at a WebmasterWorld thread.

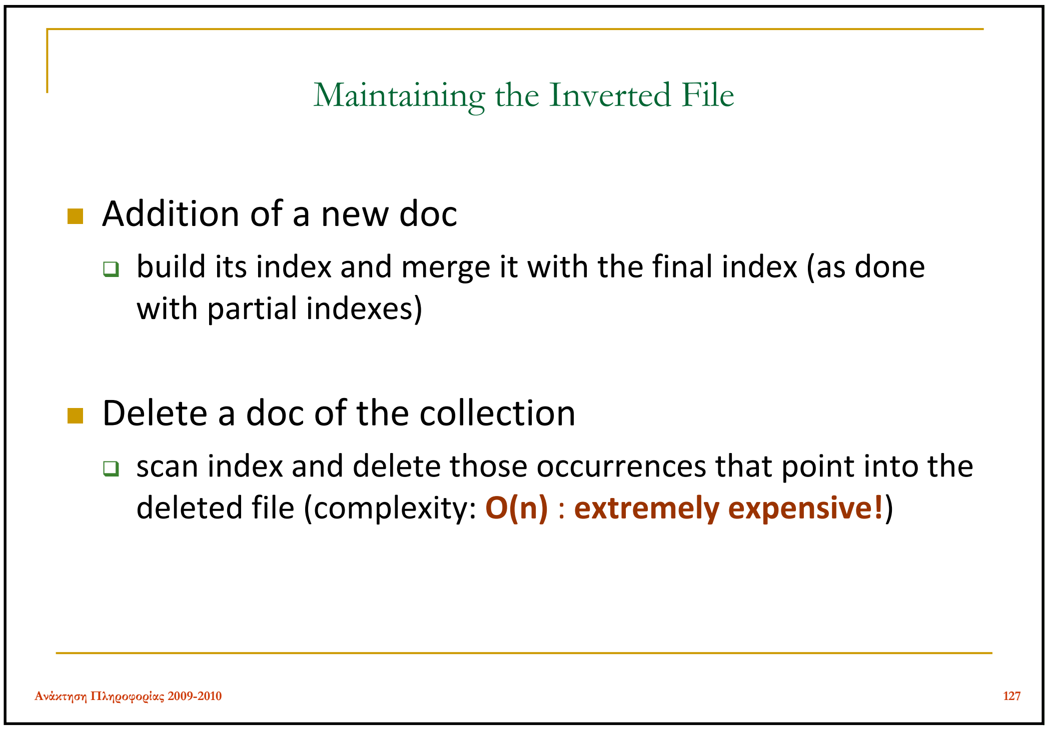
The Google Dance refers to how years ago, a change to Google's ranking algorithm often began showing up slowly across data centers as they reflected different results, a sign of coming changes. These days Google's data centers are typically always showing small changes and differences, but the differences between this data center and this one seem to be more like the extremes of the past Google Dances.

So either Google is preparing for a massive update or just messing around with our heads. As of now, these results have not yet moved over to the main Google.com results.

Other sorts of indexes

- Positional indexes
  - Same sort of sorting problem … just larger
- Building character n-gram indexes:
  - As text is parsed, enumerate *n*-grams.
  - For each *n*-gram, need pointers to all dictionary terms containing it – the "postings".
  - Note that the same "postings entry" will arise repeatedly in parsing the docs – need efficient hashing to keep track of this.
    - E.g., that the trigram *uou* occurs in the term ***deciduous*** will be discovered on each text occurrence of ***deciduous***
    - Only need to process each term once

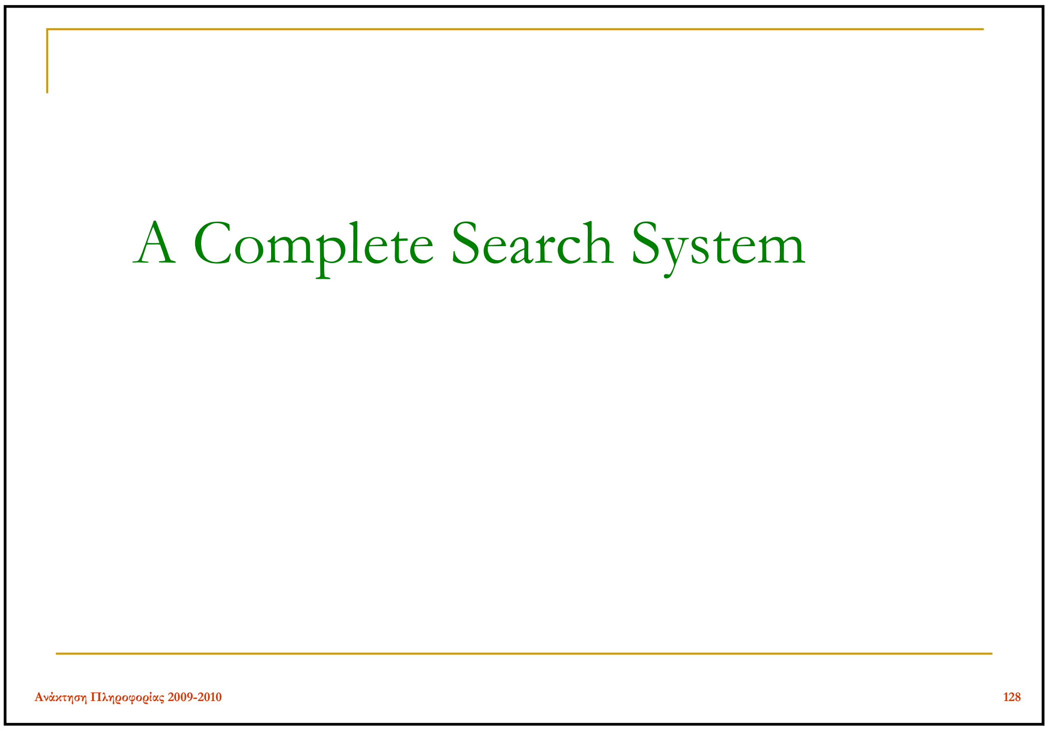
Why?



Maintaining the Inverted File

- Addition of a new doc
  - build its index and merge it with the final index (as done with partial indexes)
- Delete a doc of the collection
  - scan index and delete those occurrences that point into the deleted file (complexity: **O(n) : extremely expensive!**)

Ανάκτηση Πληροφορίας 2009-2010 127



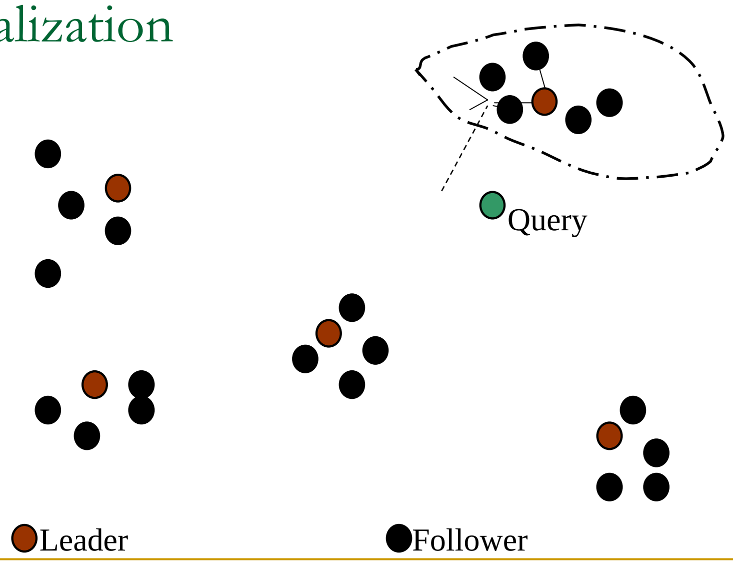
A Complete Search System

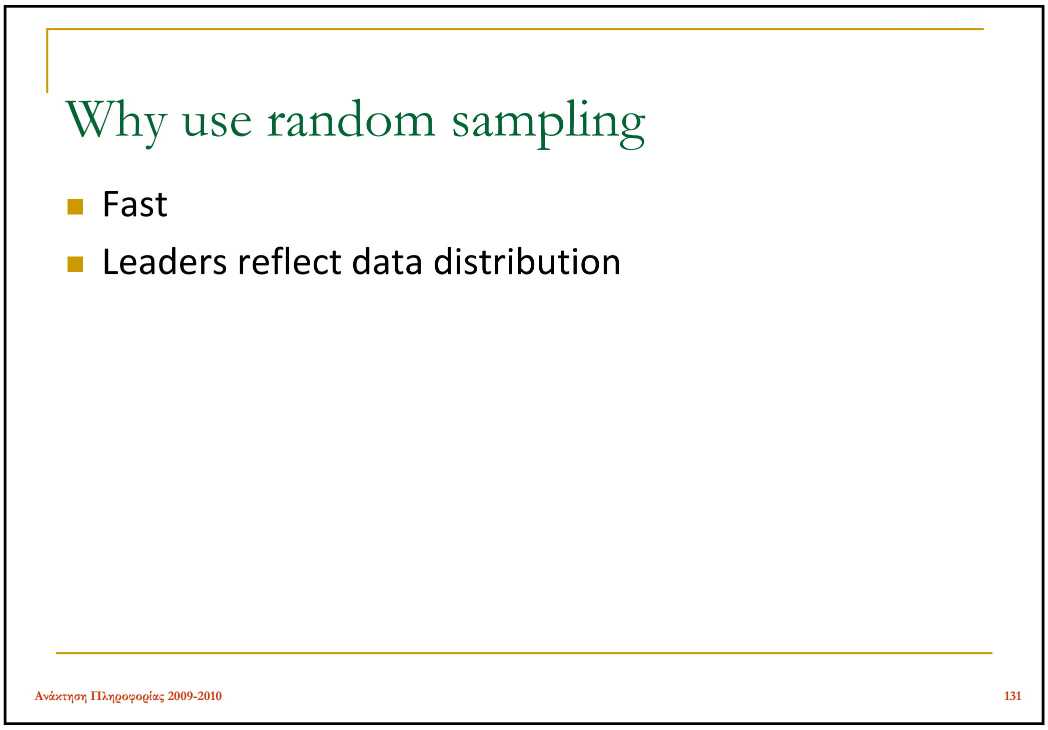
Ανάκτηση Πληροφορίας 2009-2010 128

Cluster pruning: query processing

- Process a query as follows:
  - Given query *Q*, find its nearest *leader L.*
  - Seek *K* nearest docs from among *L*'s followers.

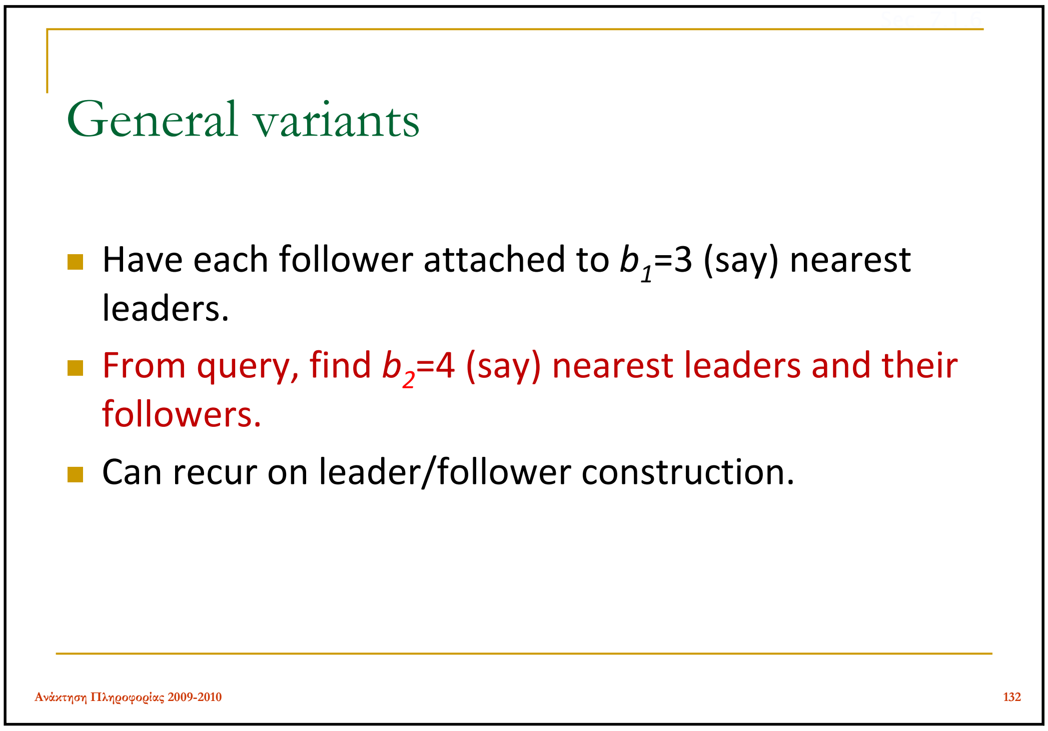
Visualization





Why use random sampling

- Fast
- Leaders reflect data distribution



General variants

- Have each follower attached to b_1=3 (say) nearest leaders.
- From query, find b_2=4 (say) nearest leaders and their followers.
- Can recur on leader/follower construction.

Top-k Aggregation

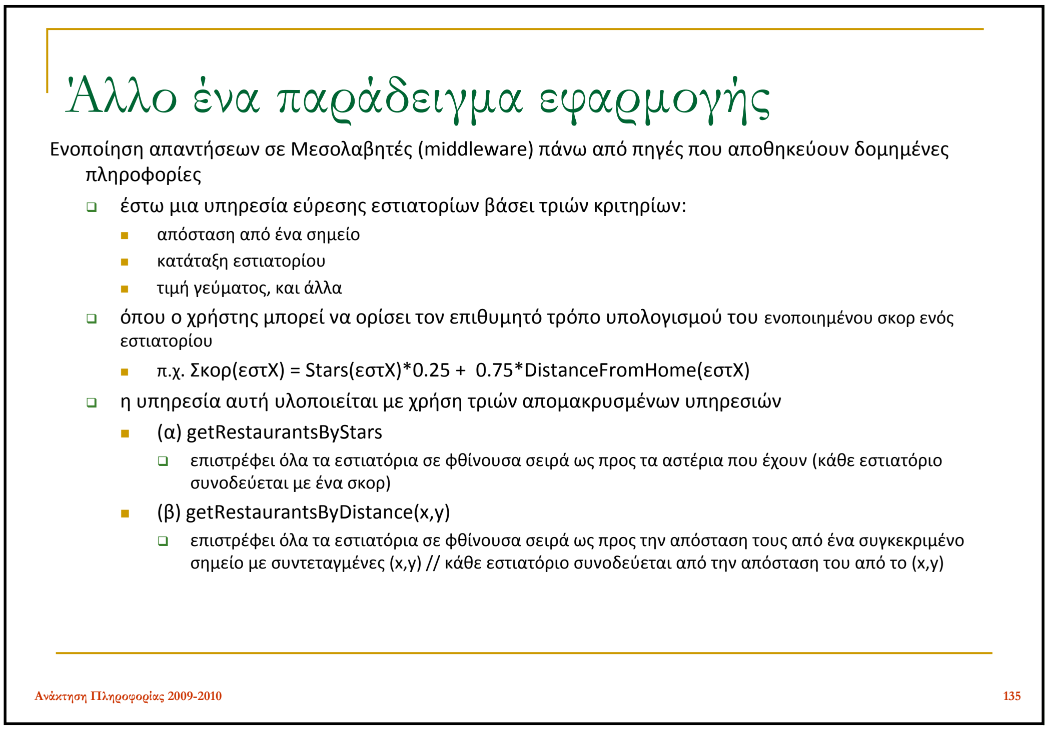
Top-k Rank Aggregation

- Έχουμε **N** αντικείμενα και τους **βαθμούς** τους βάσει **m** διαφορετικών **κριτηρίων**.
- Έχουμε έναν τρόπο να συνδυάζουμε τα m σκορ κάθε αντικειμένου σε ένα <u>ενοποιημένο σκορ</u>
  - π.χ. min, avg, sum
- Στόχος: Βρες τα **κ** αντικείμενα με το <u>υψηλότερο ενοποιημένο σκορ.</u>

Εφαρμογές:

Υπολογισμός των κορυφαίων-κ στοιχείων της απάντησης

- ενός ΣΑΠ που βασίζεται στο διανυσματικό μοντέλο (τα m κριτήρια είναι οι m όροι της επερώτησης)
- ενός μεσίτη (π.χ. μετα-μηχανής αναζήτησης) πάνω από m Συστήματα Ανάκτησης Πληροφοριών
- μιας επερώτησης σε μια Βάση Πολυμέσων
  - κριτήρια (και συνάμα χαρακτηριστικά/features): χρώμα, μορφή, υφή, …

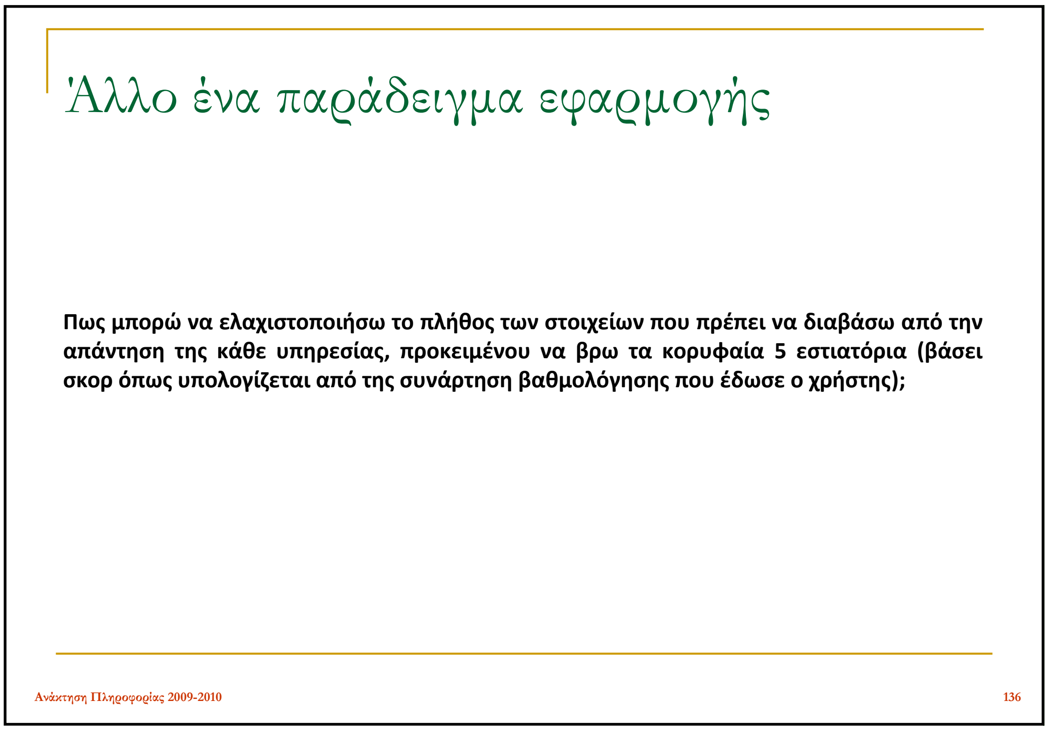


Άλλο ένα παράδειγμα εφαρμογής

Ενοποίηση απαντήσεων σε Μεσολαβητές (middleware) πάνω από πηγές που αποθηκεύουν δομημένες πληροφορίες

- έστω μια υπηρεσία εύρεσης εστιατορίων βάσει τριών κριτηρίων:
  - απόσταση από ένα σημείο
  - κατάταξη εστιατορίου
  - τιμή γεύματος, και άλλα
- όπου ο χρήστης μπορεί να ορίσει τον επιθυμητό τρόπο υπολογισμού του ενοποιημένου σκορ ενός εστιατορίου
  - π.χ. Σκορ(εστΧ) = Stars(εστΧ)*0.25 + 0.75*DistanceFromHome(εστΧ)
- η υπηρεσία αυτή υλοποιείται με χρήση τριών απομακρυσμένων υπηρεσιών
  - (α) getRestaurantsByStars
    - επιστρέφει όλα τα εστιατόρια σε φθίνουσα σειρά ως προς τα αστέρια που έχουν (κάθε εστιατόριο συνοδεύεται με ένα σκορ)
  - (β) getRestaurantsByDistance(x,y)
    - επιστρέφει όλα τα εστιατόρια σε φθίνουσα σειρά ως προς την απόσταση τους από ένα συγκεκριμένο σημείο με συντεταγμένες (x,y) // κάθε εστιατόριο συνοδεύεται από την απόσταση του από το (x,y)

Ανάκτηση Πληροφορίας 2009-2010 135



Άλλο ένα παράδειγμα εφαρμογής

Πως μπορώ να ελαχιστοποιήσω το πλήθος των στοιχείων που πρέπει να διαβάσω από την απάντηση της κάθε υπηρεσίας, προκειμένου να βρω τα κορυφαία 5 εστιατόρια (βάσει σκορ όπως υπολογίζεται από της συνάρτηση βαθμολόγησης που έδωσε ο χρήστης);

Ανάκτηση Πληροφορίας 2009-2010 136

Εύρεση των κ-κορυφαίων
Απλοϊκός Αλγόριθμος

1. Ανέκτησε <u>ολόκληρες</u> τις m λίστες
2. Υπολόγισε το ενοποιημένο σκορ του κάθε αντικειμένου
3. Ταξινόμησε τα αντικείμενα βάσει του σκορ και επέλεξε τα πρώτα κ

Παρατηρήσεις

Κόστος γραμμικό ως προς το μήκος των λιστών

Δεν αξιοποιεί το γεγονός ότι οι λίστες είναι ταξινομημένες

Εύρεση των κ-κορυφαίων
Παράδειγμα: <u>Απλοϊκός Τρόπος</u>

Έστω ότι θέλουμε να συναθροίσουμε τις διατάξεις που επιστρέφουν 3 πηγές S1, S2, S3 και ο τρόπος συνάθροισης είναι το άθροισμα.

```
S1 = < A 0.9,   C 0.8,   E 0.7,   B 0.5,   F 0.5,   G 0.5,   H 0.5 >
S2 = < B 1.0,   E 0.8,   F 0.7,   A 0.7,   C 0.5,   H 0.5,   G 0.5 >
S3 = < A 0.8,   C 0.8,   E 0.7,   B 0.5,   F 0.5,   G 0.5,   H 0.5 >
```

Ο Απλοϊκός Τρόπος

Score(**A**) = 0.9 + 0.7 + 0.8 = 2.4
Score(**B**) = 0.5 + 1.0 + 0.5 = 2
Score(**C**) = 0.8 + 0.5 + 0.8 = 2.1
Score(**E**) = 0.7 + 0.8 + 0.7 = 2.2
Score(**F**) = 0.5 + 0.7 + 0.5 = 1.7
Score(**G**) = 0.5 + 0.5 + 0.5 = 1.5
Score(**H**) = 0.5 + 0.5 + 0.5 = 1.5
Τελική διάταξη: < **A, E, C, B, F, G, H>**

Εύρεση των κ-κορυφαίων
Πιο Αποδοτικοί Αλγόριθμοι

- Γενική ιδέα: ***Άρχισε να διαβάζεις τις διατάξεις από την κορυφή. Προσπάθησε να καταλάβεις πότε πρέπει να σταματήσεις***.

- Αλγόριθμοι
  - **Fagin Algorithm (FA)** [Fagin 1999, J. CSS 58]
  - **Threshold Algorithm (TA)** [Fagin et al., PODS'2001]

Εύρεση των κ-κορυφαίων
Πιο Αποδοτικοί Αλγόριθμοι

Υποθέσεις

1. Υποθέτουμε ότι έχουμε στη διάθεση μας 2 τρόπους πρόσβασης στα αποτελέσματα μιας πηγής:

   Σειριακή πρόσβαση στις διατάξεις: φθίνουσα ως προς το σκορ

   Τυχαία προσπέλαση: Δυνατότητα εύρεσης του σκορ ενός συγκεκριμένου αντικειμένου με μία πρόσβαση

2. Συναρτήσεις βαθμολόγησης (σκορ)

   Τα σκορ ανήκουν στο διάστημα [0,1]

   Η συνάρτηση ενοποιημένου σκορ είναι **μονότονη**

   αν όλα (m) τα σκορ ενός αντικειμένου A είναι μεγαλύτερα ή ίσα των αντίστοιχων σκορ ενός αντικειμένου B, τότε σίγουρα το ενοποιημένο σκορ του A είναι μεγαλύτερο ή ίσο του σκορ του B

Εύρεση των κ-κορυφαίων
Ο Αλγόριθμος του Fagin (FA) [1999]

1.α/ Κάνε σειριακή ανάκτηση αντικειμένων από κάθε λίστα (αρχίζοντας από την κορυφή), έως η **τομή** των αντικειμένων από κάθε λίστα να έχει κ αντικείμενα

1.β/ Για κάθε αντικείμενο που ανακτήθηκε (στο 1.α) συνέλεξε τα σκορ που λείπουν (με χρήση του μηχανισμού τυχαίας προσπέλασης)

2/ Υπολόγισε το ενοποιημένο σκορ του κάθε αντικειμένου

3/ Ταξινόμησε τα αντικείμενα βάσει του ενοποιημένου σκορ και επέλεξε τα πρώτα κ

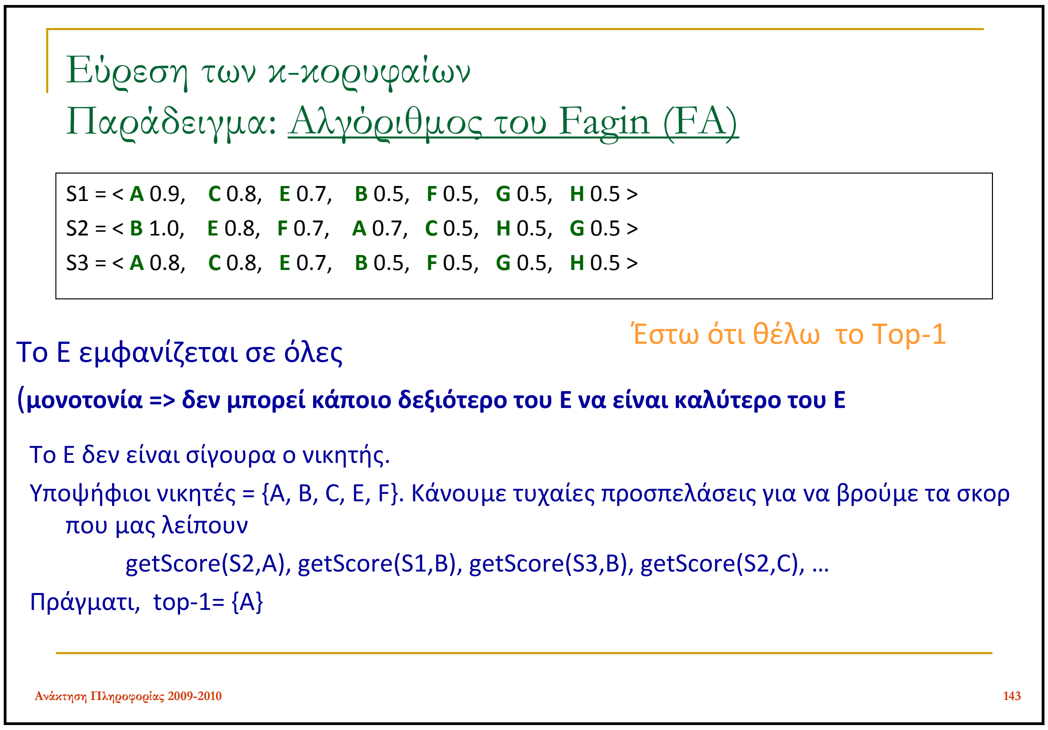
Ανάκτηση Πληροφορίας 2009-2010 141

Εύρεση των κ-κορυφαίων
Ο Αλγόριθμος του Fagin (FA) [1999]

Για οποιοδήποτε μη επιλεγμένο αντικείμενο υπάρχουν (τουλάχιστον) κ που είναι καλύτερα από αυτό

Σχόλια
Αξιοποιεί (α) το γεγονός ότι οι λίστες είναι ταξινομημένες και (β) ότι η συνάρτηση ενοποίησης είναι μονότονη
[-] Το πλήθος των αντικειμένων που θα ανακτηθούν μπορεί να είναι μεγάλο

Ανάκτηση Πληροφορίας 2009-2010 142



Εύρεση των κ-κορυφαίων
Παράδειγμα: Αλγόριθμος του Fagin (FA)

S1 = < **A** 0.9, **C** 0.8, **E** 0.7, **B** 0.5, **F** 0.5, **G** 0.5, **H** 0.5 >
S2 = < **B** 1.0, **E** 0.8, **F** 0.7, **A** 0.7, **C** 0.5, **H** 0.5, **G** 0.5 >
S3 = < **A** 0.8, **C** 0.8, **E** 0.7, **B** 0.5, **F** 0.5, **G** 0.5, **H** 0.5 >

Έστω ότι θέλω το Top-1

Το Ε εμφανίζεται σε όλες

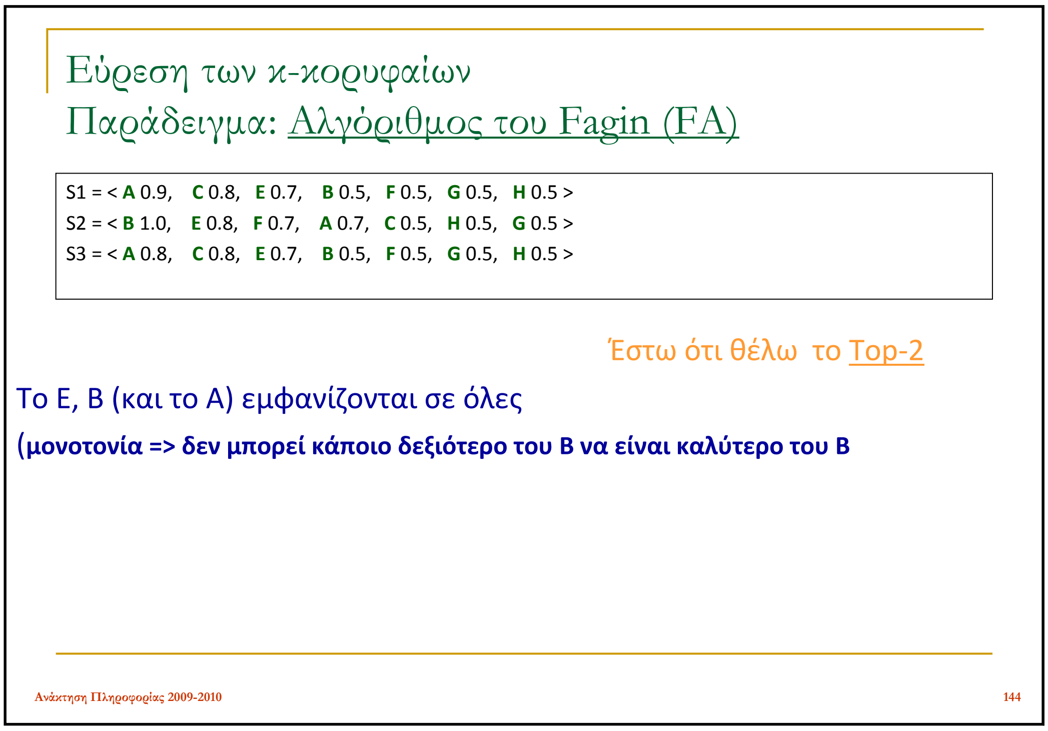
(μονοτονία => δεν μπορεί κάποιο δεξιότερο του Ε να είναι καλύτερο του Ε

Το Ε δεν είναι σίγουρα ο νικητής.

Υποψήφιοι νικητές = {A, B, C, E, F}. Κάνουμε τυχαίες προσπελάσεις για να βρούμε τα σκορ που μας λείπουν

getScore(S2,A), getScore(S1,B), getScore(S3,B), getScore(S2,C), …

Πράγματι, top-1= {A}



Εύρεση των κ-κορυφαίων
Παράδειγμα: Αλγόριθμος του Fagin (FA)

S1 = < **A** 0.9, **C** 0.8, **E** 0.7, **B** 0.5, **F** 0.5, **G** 0.5, **H** 0.5 >
S2 = < **B** 1.0, **E** 0.8, **F** 0.7, **A** 0.7, **C** 0.5, **H** 0.5, **G** 0.5 >
S3 = < **A** 0.8, **C** 0.8, **E** 0.7, **B** 0.5, **F** 0.5, **G** 0.5, **H** 0.5 >

Έστω ότι θέλω το Top-2

Το Ε, Β (και το Α) εμφανίζονται σε όλες

(μονοτονία => δεν μπορεί κάποιο δεξιότερο του Β να είναι καλύτερο του Β

Εύρεση των κ-κορυφαίων

Ο Αλγόριθμος TA (Threshold Algorithm) [Fagin et al. 2001]

Ιδέα:

Υπολόγισε το μέγιστο σκορ που μπορεί να έχει ένα αντικείμενο που δεν έχουμε συναντήσει ακόμα.

1/ Κάνε σειριακή ανάκτηση αντικειμένων από κάθε λίστα (αρχίζοντας από την κορυφή) και με χρήση *τυχαίας προσπέλασης* βρες όλα τα σκορ κάθε αντικειμένου

2/ Ταξινόμησε τα αντικείμενα (βάσει του ενοποιημένου σκορ) και κράτησε τα καλύτερα κ

3/ Σταμάτησε την σειριακή ανάκτηση όταν **τα σκορ των παραπάνω κ αντικειμένων δεν μπορεί να είναι μικρότερα του μέγιστου πιθανού σκορ των απαρατήρητων αντικειμένων** (threshold).

Εύρεση των κ-κορυφαίων

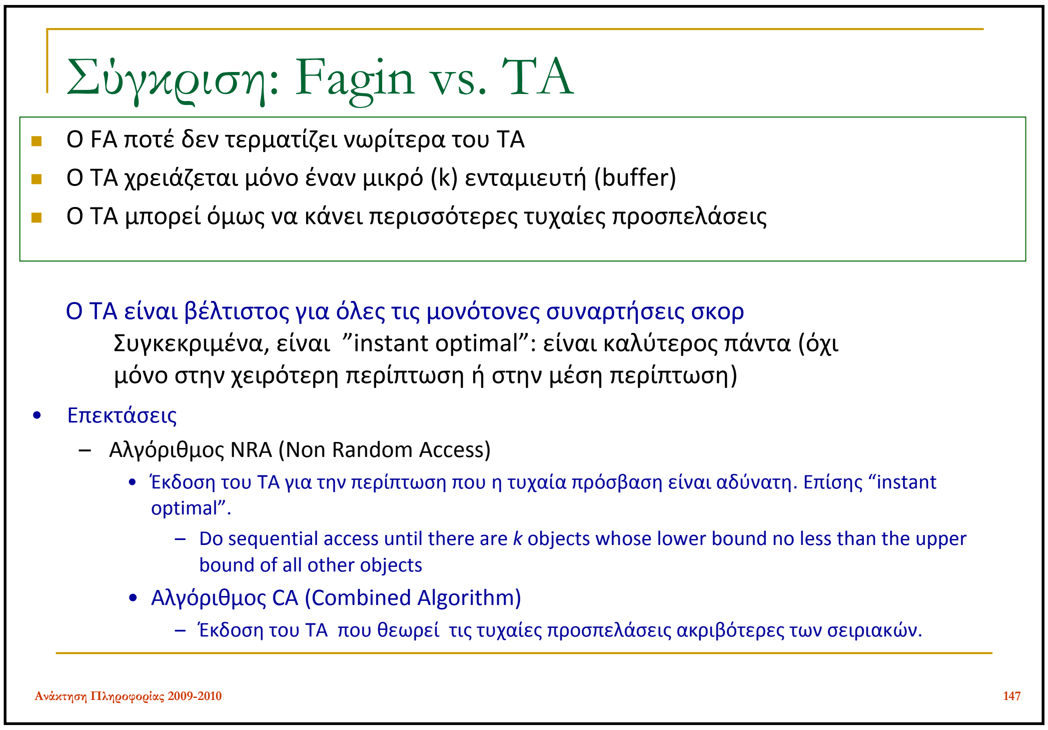
Παράδειγμα: Αλγόριθμος TA:

S1 = < **A** 0.9, **C** 0.8, **E** 0.7, **B** 0.5, **F** 0.5, **G** 0.5, **H** 0.5 >
S2 = < **B** 1.0, **E** 0.8, **F** 0.7, **A** 0.7, **C** 0.5, **H** 0.5, **G** 0.5 >
S3 = < **A** 0.8, **C** 0.8, **E** 0.7, **B** 0.5, **F** 0.5, **G** 0.5, **H** 0.5 >

Έστω ότι θέλω το Top-1

Score(A) = 0.9 + 0.7 + 0.8 = 2.4
Score(B) = 0.5 + 1.0 + 0.5 = 2
UpperBound = 0.9 + 1.0 + 0.8 = 2.7
αφού 2.7 > 2.4 συνεχίζω

Score(C) = 0.8 + 0.5 + 0.8 = 2.1
Score(E) = 0.7 + 0.8 + 0.7 = 2.2
UpperBound = 0.8 + 0.8 + 0.8 = 2.4
αφού 2.4 δεν είναι μεγαλύτερο του 2.4 (σκορ του A) σταματάω.



Σύγκριση: Fagin vs. TA

- Ο FA ποτέ δεν τερματίζει νωρίτερα του TA
- Ο TA χρειάζεται μόνο έναν μικρό (k) ενταμιευτή (buffer)
- Ο TA μπορεί όμως να κάνει περισσότερες τυχαίες προσπελάσεις

Ο TA είναι βέλτιστος για όλες τις μονότονες συναρτήσεις σκορ
Συγκεκριμένα, είναι ”instant optimal”: είναι καλύτερος πάντα (όχι μόνο στην χειρότερη περίπτωση ή στην μέση περίπτωση)

- Επεκτάσεις
  - Αλγόριθμος NRA (Non Random Access)
    - Έκδοση του TA για την περίπτωση που η τυχαία πρόσβαση είναι αδύνατη. Επίσης “instant optimal”.
      - Do sequential access until there are *k* objects whose lower bound no less than the upper bound of all other objects
    - Αλγόριθμος CA (Combined Algorithm)
      - Έκδοση του TA που θεωρεί τις τυχαίες προσπελάσεις ακριβότερες των σειριακών.

Ανάκτηση Πληροφορίας 2009-2010 147