Πανεπιστήμιο Κρήτης, Τμήμα Επιστήμης Υπολογιστών
Άνοιξη 2009

HY463 - Συστήματα Ανάκτησης Πληροφοριών
Information Retrieval (IR) Systems

Ευρετηρίαση, Αποθήκευση και Οργάνωση Αρχείων (Indexing, Storage and File Organization)

Κεφάλαιο 8

Δομές Ευρετηρίου: Διάρθρωση Διάλεξης

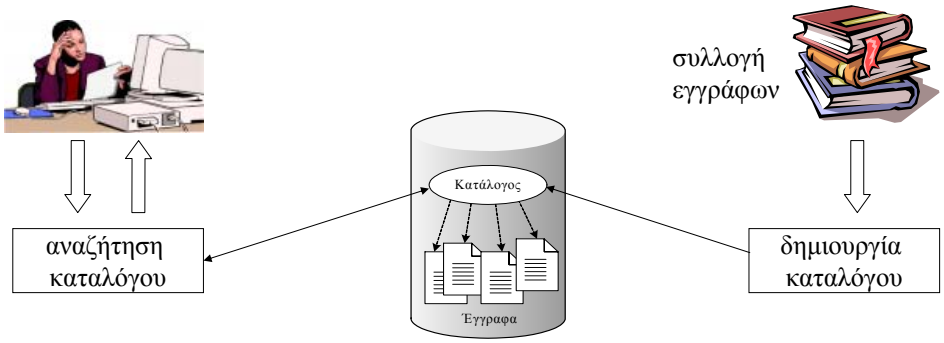
- Εισαγωγή – κίνητρο
- Ανεστραμμένα Αρχεία (Inverted files)
- Αρχεία Υπογραφών (Signature files)
- Δένδρα Καταλήξεων (Suffix trees)

Ευρετηριασμός Κειμένου:Εισαγωγή

- Σκοπός
  - **Σχεδιασμός δομών δεδομένων που επιτρέπουν την αποδοτική υλοποίηση της γλώσσας επερώτησης**
- Απλοϊκή προσέγγιση: σειριακή αναζήτηση (online sequential search)
  - Ικανοποιητική μόνο αν η συλλογή των κειμένων είναι **μικρή**
  - Είναι η **μόνη** επιλογή αν η συλλογή κειμένων είναι **ευμετάβλητη**
  - **Σχεδιασμός δομών δεδομένων, που ονομάζονται ευρετήρια (called *indices*), για επιτάχυνση της αναζήτησης**

Χρήση Καταλόγων/Ευρετηρίων

Τα συστήματα ανάκτησης σπάνια αναζητούν την πληροφορία απευθείας στη συλλογή εγγράφων. Συνήθως, χρησιμοποιούνται κατάλογοι οι οποίοι επιταχύνουν τη διαδικασία αναζήτησης.



Ανάγκες Γλωσσών Επερώτησης (και μοντέλων ανάκτησης γενικότερα)

- Απλές
  - βρες έγγραφα που **περιέχουν** μια λέξη t
  - βρες **πόσες φορές** εμφανίζεται η λέξη t σε ένα έγγραφο
  - βρες τις **θέσεις** των εμφανίσεων της λέξης t στο έγγραφο
- Πιο σύνθετες
  - λογικές (Boolean) επερωτήσεις
  - επερωτήσεις εγγύτητας (phrase/proximity queries)
  - ταιριάσματος προτύπου (pattern matching)
  - κανονικές εκφράσεις (regular expressions)
  - δομικές επερωτήσεις (structure-based queries)
  - ...

Σχεδιάζουμε το ευρετήριο ανάλογα με το μοντέλο ανάκτησης και τη γλώσσα επερώτησης

Γενική (Λογική) μορφή ενός ευρετηρίου

Indexing Items

Documents	k_1	k_2	...	k_j	...	k_t
d_1	$c_{1,1}$	$c_{2,1}$	...	$c_{i,1}$	...	$c_{t,1}$
d_2	$c_{1,2}$	$c_{2,2}$	...	$c_{i,2}$	...	$c_{t,2}$
...	...	...	...	...	...	...
d_i	$c_{1,j}$	$c_{2,j}$	...	$c_{i,j}$	...	$c_{t,j}$
...	...	...	...	...	...	...
d_N	$c_{1,N}$	$c_{2,N}$	...	$c_{i,N}$	...	$c_{t,N}$

c_{ij}: το κελί που αντιστοιχεί στο έγγραφο d_i και στον όρο k_j, το οποίο μπορεί να περιέχει:

- ένα w_{ij} που να δηλώνει την παρουσία ή απουσία του k_j στο d_i (ή τη σπουδαιότητα του k_j στο d_i)
- τις θέσεις στις οποίες ο όρος k_j εμφανίζεται στο d_i (αν πράγματι εμφανίζεται)

Ερωτήματα:

Τι πρέπει να έχει το κάθε c_{ij}

Πώς να υλοποιήσουμε αυτή τη λογική δομή ώστε να έχουμε καλή απόδοση;

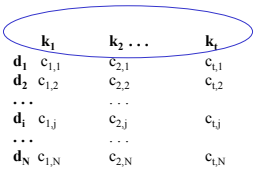
Τεχνικές Ευρετηριασμού (Indexing Techniques)

- Ανεστραμμένα Αρχεία (Inverted files)
  - η πιο διαδεδομένη τεχνική
- Δένδρα και Πίνακες Καταλήξεων (Suffix trees and arrays)
  - γρήγορες για “phrase queries” αλλά η κατασκευή και η συντήρησή τους είναι δυσκολότερη και ακριβότερη
- Αρχεία Υπογραφών (Signature files)
  - Χρησιμοποιήθηκαν πολύ τη δεκαετία του 80. Σπανιότερα σήμερα – αλλά σε κατανεμημένα διάφορες παραλλαγές τους.

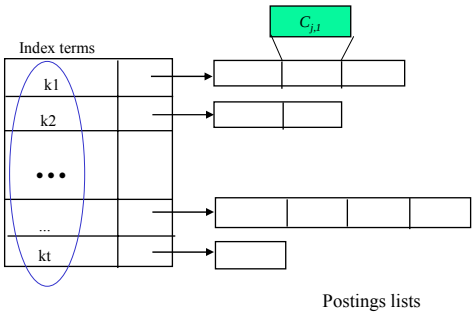
Ανεστραμμένα Αρχεία (Inverted Files)

Ανεστραμμένο Αρχείο

Λογική Μορφή Ευρετηρίου



Μορφή Ανεστραμμένου Ευρετηρίου



Άρα δεν δεσμεύουμε χώρο για τα «μηδενικά κελιά» της λογικής μορφής του ευρετηρίου

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**

Ανεστραμμένο αρχείο για ένα μόνο έγγραφο και αποθήκευση θέσεων εμφάνισης κάθε λέξης

Κείμενο

That house has a garden. The garden has many flowers. The flowers are beautiful

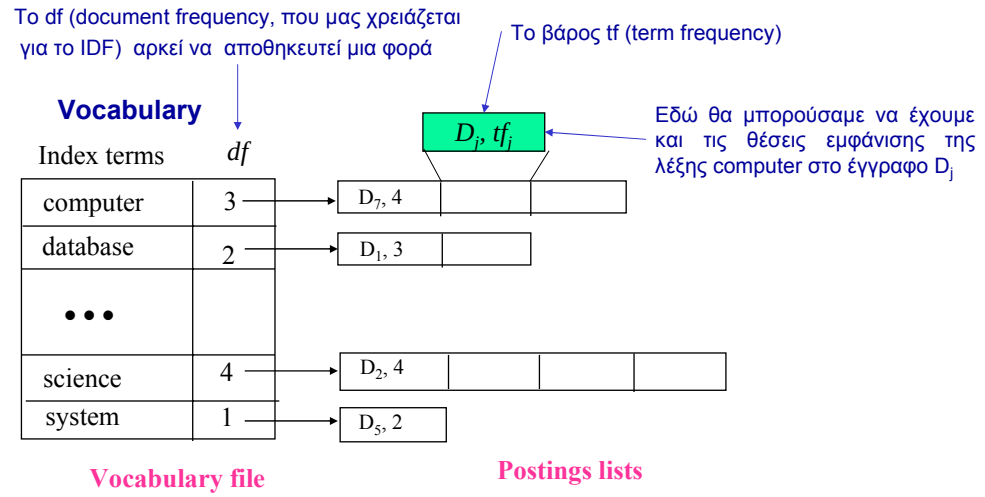
1 6 12 16 18 25 29 36 40 45 54 58 66 70

Inverted File:

Vocabulary	Occurrences
beautiful	70
flowers	45, 58
garden	18, 29
house	6

Τι άλλο θα κάνατε (κρατούσατε) αν είχαμε πολλά έγγραφα και θέλαμε να υλοποιήσουμε το Διανυσματικό Μοντέλο;

Ανεστραμμένο αρχείο για πολλά έγγραφα, και βάρυνση tf-idf



Παράδειγμα ανεστραμμένου αρχείου όπου για κάθε λέξη i και έγγραφο j κρατάμε μόνο το $freq_{ij}$

Document Corpus

Doc	Text
1	Pease porridge hot
2	Pease porridge cold
3	Pease porridge in the pot
4	Pease porridge hot, pease porridge not cold
5	Pease porridge cold, pease porridge not hot
6	Pease porridge hot in the pot

Inverted File

Vocabulary	Inverted Lists
cold	<2,1> <4,1> <5,1>
hot	<1,1> <4,1> <5,1> <6,1>
in	<3,1> <6,1>
not	<4,1> <5,1>
pease	<1,1> <2,1> <3,1> <4,2> <5,2> <6,1>
porridge	<1,1> <2,1> <3,1> <4,2> <5,2> <6,1>
pot	<3,1> <6,1>
the	<3,1> <6,1>

Another example

	term	df	document ids
1	Algorithms	3	: 3 5 7
2	Application	2	: 3 17
3	Delay	2	: 11 12
4	Differential	8	: 4 8 10 11 12 13 14 15
5	Equations	10	: 1 2 4 8 10 11 12 13 14 15
6	Implementation	2	: 3 7
7	Integral	2	: 16 17
8	Introduction	2	: 5 6
9	Methods	2	:
10	Nonlinear	2	: 9 13
11	Ordinary	2	: 8 10
12	Oscillation	2	: 11 12
13	Partial	2	: 4 13
14	Problem	2	: 6 7
15	Systems	3	: 6 8 9
16	Theory	4	: 3 11 12 17

Block Addressing

- The text is divided in blocks
- The occurrences point to the blocks where the word appears

Block Addressing: Example

That house has a garden. The garden has many flowers. The flowers are beautiful

1 6 12 16 18 25 29 36 40 45 54 58 66 70

Vocabulary	Occurrences
beautiful	70
flowers	45, 58
garden	18, 29
house	6

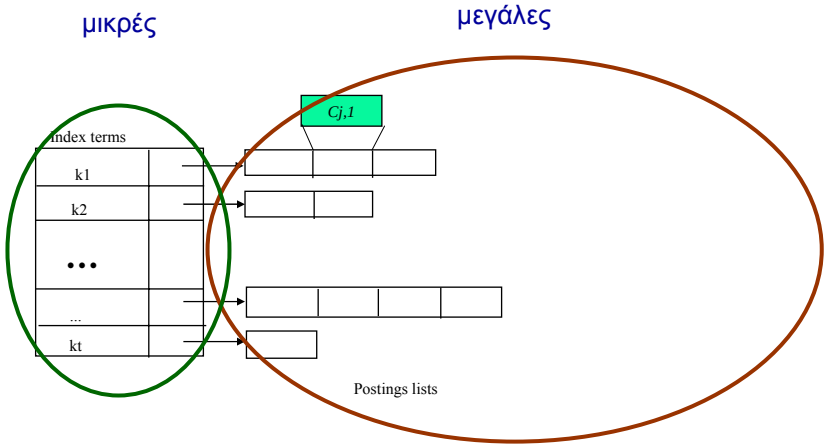
Block 1	Block 2	Block 3	Block 4
That house has a	garden. The garden has	many flowers. The flowers	are beautiful

Vocabulary	Occurrences
beautiful	4
flowers	3
garden	2
house	1

Block Addressing

- Advantages:
  - the number of pointers is smaller than positions
  - all the occurrences of a word inside a single block are collapsed to one reference
  - (**indices of only 5% overhead over the text size can be obtained with this technique. Of course this depends on the block size**).
    - In many cases instead of defining the block size, we define the number of blocks (in this way we know how many bits we need per pointer)
- Disadvantages:
  - online sequential search over the qualifying blocks if exact positions are required
    - e.g. for finding the sentence where the word occurs
    - e.g. for evaluating a context (phrasal or proximity) query

Ανεστραμμένα Αρχεία: Απαιτήσεις Χώρου



Ανεστραμμένα Αρχεία: Απαιτήσεις Χώρου

Notations
- *n*: the size of the text (of all documents in the collection)
- V: the size of the vocabulary

For the Vocabulary:
- Rather **small**.
- According to *Heaps'* law the vocabulary grows as $O(n^{\beta})$, where β is a constant between 0.4 and 0.6 in practice. So V ~ sqrt(n) // άρα ανάλογο της τετραγωνικής ρίζας του μεγέθους της συλλογής)

For Occurrences:
- **Much more** space.
- Since each word appearing in the text is referenced once in that structure (i.e. we keep a pointer), the extra space is *O(n)*
- To reduce space requirements, a technique called *block addressing* is used

how?

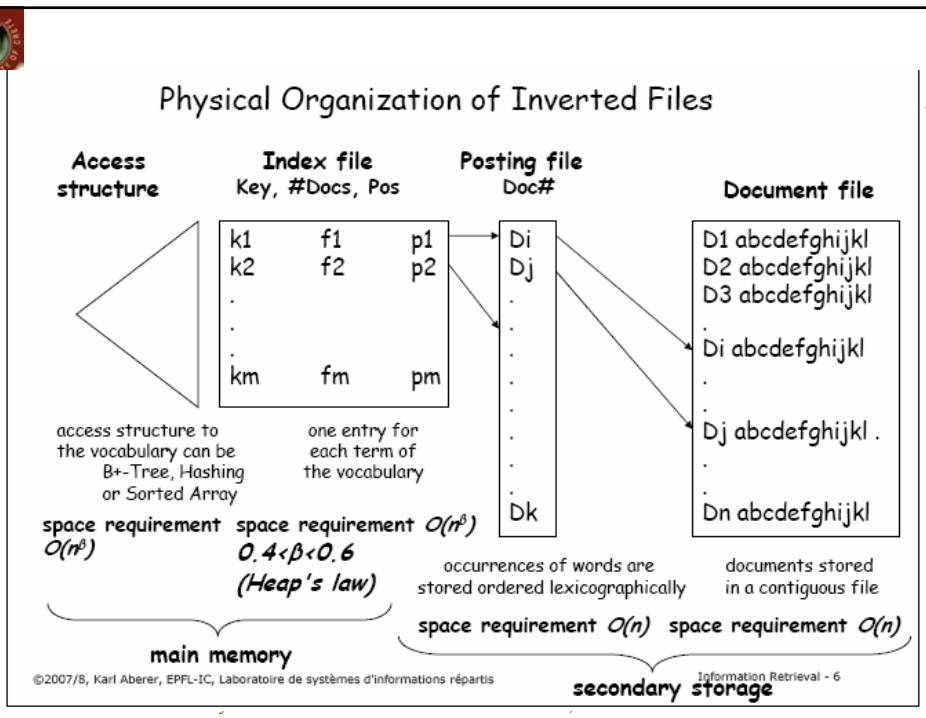
Size of Inverted Files as percentage of the size of the whole collection

45% of all words are stopwods

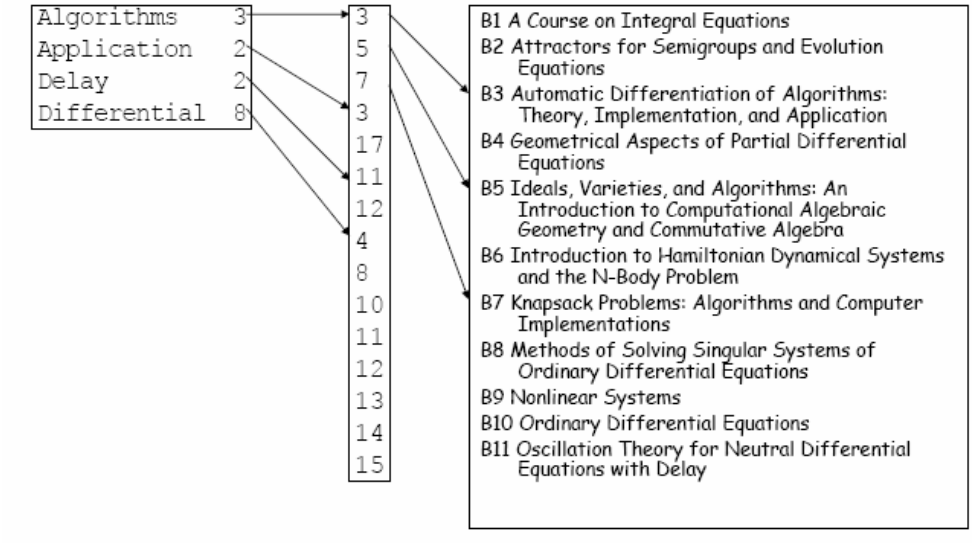
Index	Small collection (1Mb)		Medium collection (200Mb)		Large collection (2Gb)	
Addressing words	**45%**	**73%**	**36%**	**64%**	**35%**	**63%**
Addressing 64K blocks	**27%**	**41%**	**18%**	**32%**	**5%**	**9%**
Addressing 256 blocks	**18%**	**25%**	**1.7%**	**2.4%**	**0.5%**	**0.7%**
	Without stopwords	All words	Without stopwords	All words	Without stopwords	All words

Addressing words: 4 bytes per pointer (2^32 ~ giga)
Addressing 64K blocks: 2 bytes per pointer
Addressing 256 blocks: 1 byte per pointer

Searching an inverted index



(cont)



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
- If block addressing is used we have to search the text of the blocks in order to get the exact positions and number of occurrences

1/ Vocabulary search

As Searching task on an inverted file always starts in the vocabulary, it is better to **store the vocabulary in a separate file**

- this file is not so big so it is possible to keep it at main memory at search time

Suppose we want to search for a word of length *m.*

The structures most used to store the vocabulary are ***hashing, tries*** or ***B-trees.***

Options:

- *Cost of searching a sequential file: O(V)*
- *Cost of searching assuming hashing: O(m)*
- *Cost of searching assuming tries: O(m)*
- *Cost of searching assuming the file is ordered (lexicographically): O(log V)*
  - this option is cheaper in space and very competitive

Υπόβαθρο/Επανάληψη: **Tries**

Tries

- multiway trees for stroring strings
- able to retrieve any string in time proportional to its length (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)

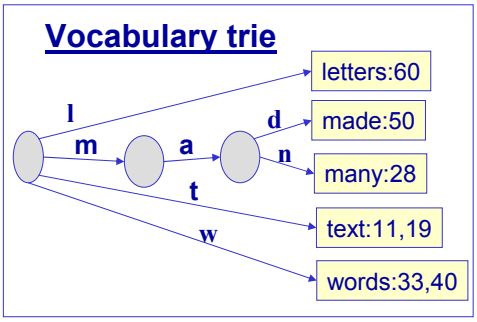
Tries: Παράδειγμα

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**.

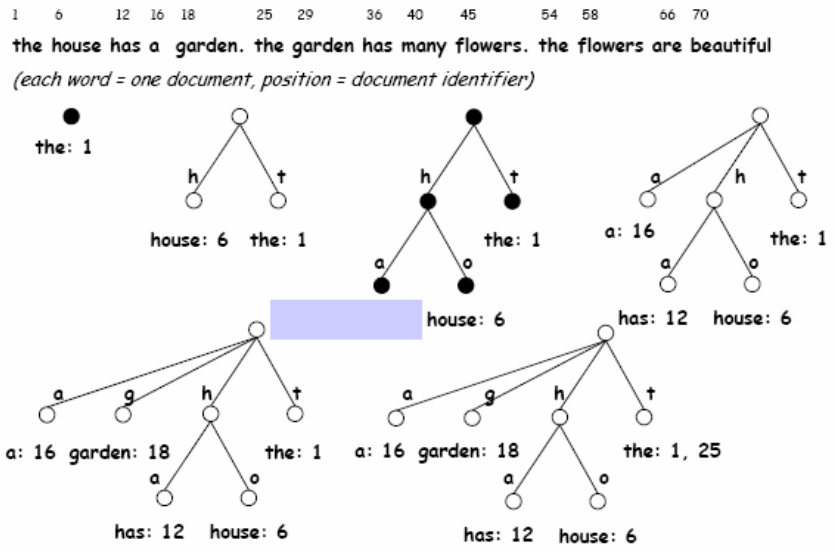
Vocabulary
text (11)
text (19)
many (28)
words (33)
words (40)
made (50)
letters (60)

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/ Vocabulary Search (II)

Remarks

- prefix and range queries
  - can also be solved with binary search, tries or B-trees but not with hashing
- context queries
  - are more difficult to solve with inverted indices
    1. each element must be searched separately and
    2. a list (in increasing positional order) is generated for each one
    3. The lists of all elements are traversed in synchronization to find places where all the words appear in sequence (for a phrase) or appear close enough (for proximity).

Inverted Index: A general remark

Experiments show that both the space requirements and the amount of text traversed can be close to O(n^0.85). Hence, inverted indices allow us to have sublinear search time and sublinear space requirements. This is not possible on other indices.

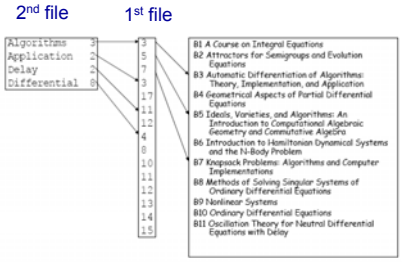
Constructing an Inverted File

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 (II)

- Once the text is exhausted the vocabulary is written to disk with the list of occurrences. Two files are created:
  - in the first file, the list of occurrences are stored contiguously
  - in the second file, 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

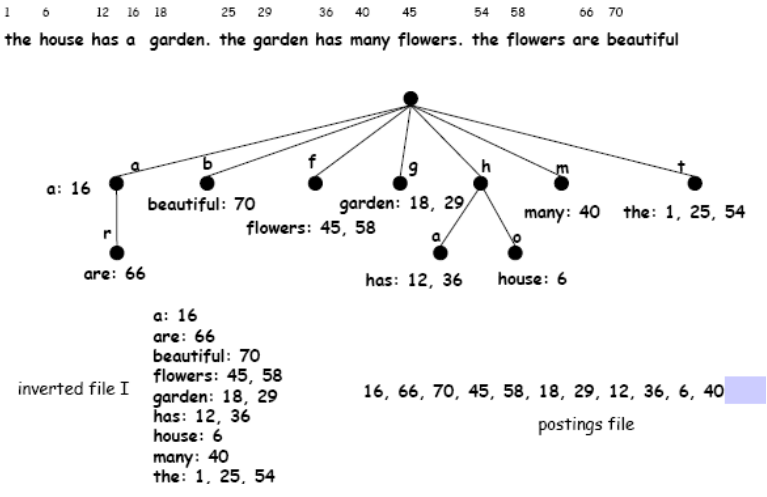


Trie: O(1) per text character

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)



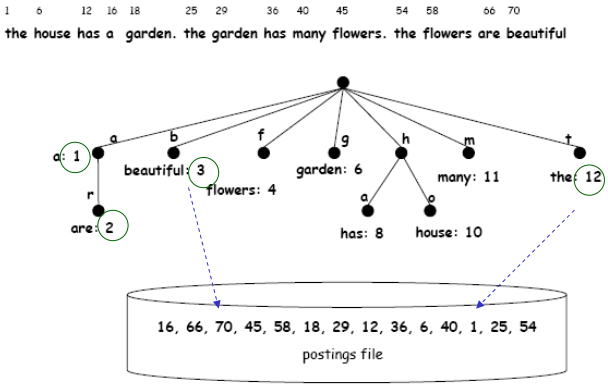
Once the complete trie structure is constructed the inverted file can be derived from it:

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.



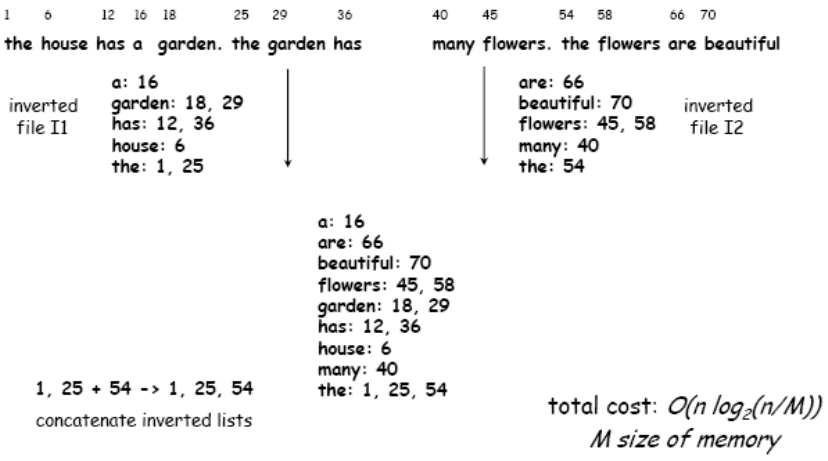
What if the Inverted Index does not fit in main memory ?

- 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 $\mathbf{I_i}$** 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 $\mathbf{I_i}$ exist on disk
- The partial indices are **merged** to obtain the final index

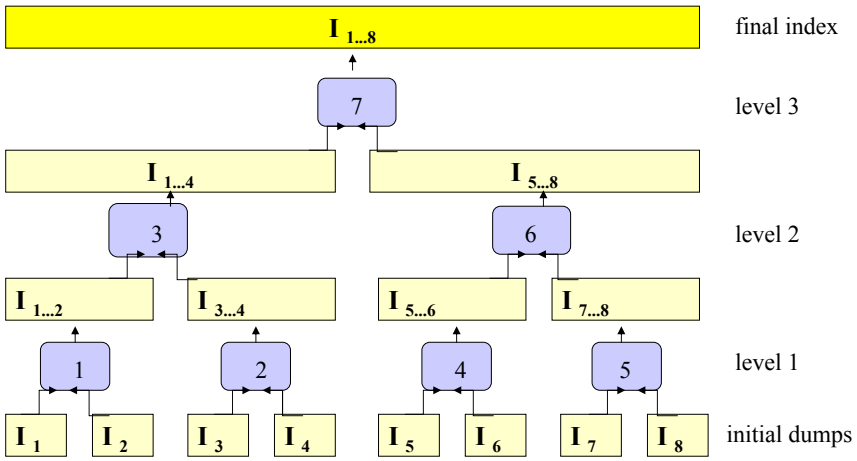
Merging two partial indices I1 and I2

- 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



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))*

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!**)

Αποτίμηση Boolean επερωτήσεων με χρήση ανεστραμμένων αρχείων

Αποτίμηση με χρήση ανεστραμμένων αρχείων

- Primitive keyword: Retrieve containing documents using the inverted index.
- OR: Recursively retrieve e_1 and e_2 and take union of results.
- AND: Recursively retrieve e_1 and e_2 and take intersection of results.
- BUT: Recursively retrieve e_1 and e_2 and take set difference of results.

Evaluating Phrasal and Proximity Queries with Inverted Indices

- Phrasal Queries
  - Must have an inverted index that also stores *positions* of each keyword in a document.
  - Retrieve documents and positions for each individual word, **intersect** documents, and then finally **check** for ordered contiguity of keyword positions.

    Best to start contiguity check with the *least common word* in the phrase.

- Proximity Queries
  - Use approach similar to phrasal search to find documents in which all keywords are found in a context that satisfies the proximity constraints -- a list (in increasing positional order) is generated for each one
  - The lists of all elements are traversed in synchronization to find places where all the words appear close enough (for proximity).
  - During binary search for positions of remaining keywords, find closest position of k_i to p and check that it is within maximum allowed distance.

Inverted Index: Κατακλείδα

- Is probably the most adequate indexing technique
- Appropriate when the text collection is large and semi-static
- If the text collection is volatile online searching is the only option
- Some techniques combine online and indexed searching

Είδαμε τρόπους για να **μειώσουμε το μέγεθος** ενός ανεστραμμένου ευρετηρίου (λέξεις αποκλεισμού, block addressing). Θα δούμε και άλλους τρόπους στο μάθημα περί συμπίεσης

(συγκεκριμένα τρόπους μείωσης του χώρου που καταλαμβάνουν οι λίστες εμφανίσεων)

Δομές Ευρετηρίου: Διάρθρωση Διάλεξης

- Εισαγωγή – κίνητρο
- Ανεστραμμένα Αρχεία (Inverted files)
- Αρχεία Υπογραφών (Signature files)
- Δένδρα Καταλήξεων (Suffix trees)

Signature files (αρχεία υπογραφών)

Αρχεία Υπογραφών (Signature Files)

Κύρια σημεία:

- Δομή ευρετηρίου που βασίζεται στο **hashing**
- Μικρή χωρική επιβάρυνση (**10%-20%** του μεγέθους των κειμένων)
- Χαρακτηρίζεται από απώλεια πληροφορίας: κατά τη διαδικασία της αναζήτησης σχετικών εγγράφων μπορεί να ανακτηθούν έγγραφα τα οποία δεν περιέχουν τους όρους του ερωτήματος.

- Αναζήτηση = σειριακή αναζήτηση στο αρχείο υπογραφών
- Κατάλληλη για όχι πολύ μεγάλα κείμενα

Αρχεία Υπογραφών

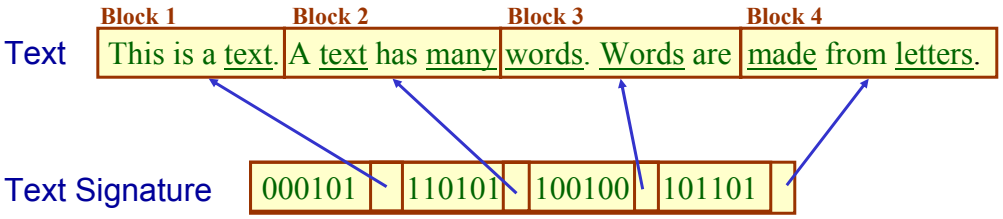
Συγκεκριμένα

1. Χρήση hash function που αντιστοιχεί λέξεις κειμένου σε bit masks των B bits

2. **Διαμέριση** του κειμένου σε blocks των b λέξεων το καθένα
   1. Bit mask of a block = **Bitwise OR** of the bits masks of all words in the block
   2. Bit masks are then concatenated

Αρχεία Υπογραφών: Παράδειγμα

b = 3 (3 words per block) **B = 6** (bit masks of 6 bits)



Signature Function

h(text)= 000101
h(many)= 110000
h(words)=100100
h(made)= 001100
h(letters)=100001

Γιατί Bitwise-OR?

Αρχεία Υπογραφών: Αναζήτηση

Έστω ότι αναζητούμε μια λέξη w:

1/ W := h(w) (we hash the word to a bit mask W)

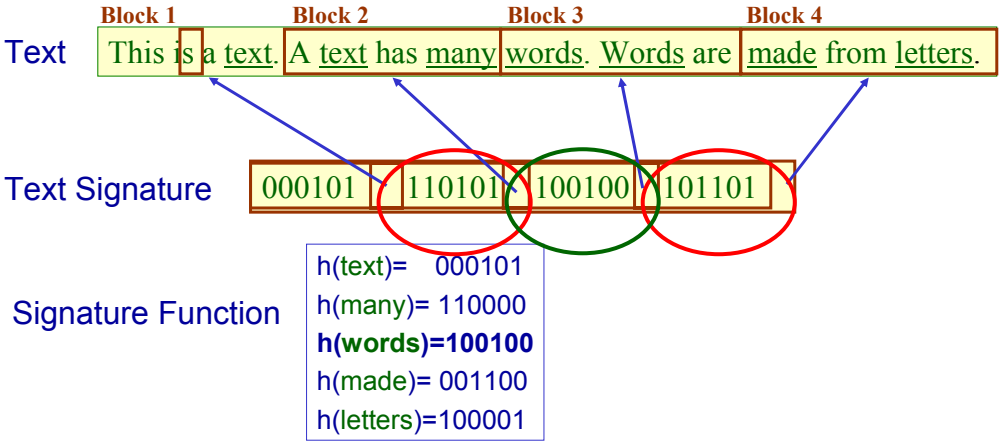
2/ Compare W with all bit masks Bi of all text blocks
If (W **&** Bi = W), the text block i is candidate (may contain the word w)

3/ For all candidate text blocks, perform an online traversal to verify that the word w is actually there

False drops (false hits)

False drop (false hit, false positive): All bits of the W are set in Bi but the word w is not there

w=«words», h(«words»)=100100



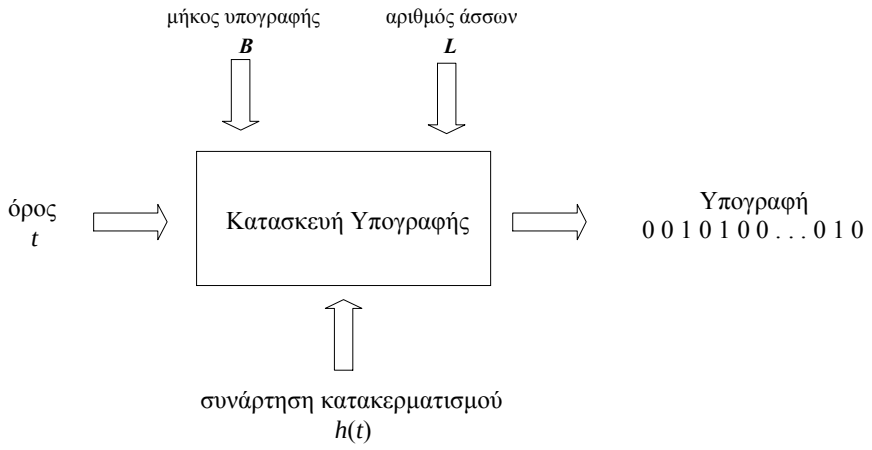
Υπογραφές (σύγκρουση)

- Ακόμα και να δεν υπάρχει υπέρθεση (ORing) υπάρχει περίπτωση δύο διαφορετικοί όροι να έχουν **την ίδια υπογραφή**.
- Το φαινόμενο αυτό καλείται **σύγκρουση** (collision) και η εμφάνισή του επηρεάζεται από το μέγεθος της υπογραφής και από τη συνάρτηση κατακερματισμού που χρησιμοποιείται.
- Όσο αυξάνει ο αριθμός των συγκρούσεων αυξάνει και ο αριθμός των false alarms (drops).

Διαμόρφωση (Configuration) υπογραφών

- Σχεδιαστικοί στόχοι:
  - **Μείωσε την πιθανότητα** εμφάνισης **false drops**
  - Κράτησε το **μέγεθος** του αρχείου υπογραφών **μικρό**
    - δεν έχουμε κανένα false drop αν b=1 και $B=\log_2(V)$
- Παράμετροι:
  - B (το μέγεθος των bit mask)
  - L (L<B) το πλήθος των bit που είναι 1 (σε κάθε h(w))
- The (space)-(false drop probability) tradeoff:
  - 10% space overhead => 2% false drop probability
  - 20% space overhead => 0.046% false drop probability

Εξαγωγή Υπογραφής



Αρχεία Υπογραφών: Άλλες Παρατηρήσεις

- Μέγεθος αρχείου υπογραφών:
  - bit masks of each block plus one pointer for each block
    - (pointing to the corresponding position at the original text)
- Συντήρηση αρχείου υπογραφών:
  - Η προσθήκη/διαγραφή αρχείων αντιμετωπίζεται εύκολα
    - προσθέτονται/διαγράφονται τα αντίστοιχα bit masks

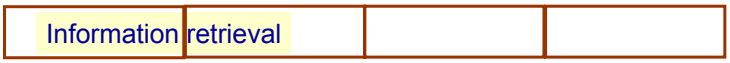
Signature files: Phrase and Proximity Queries

- Good for **phrase searches** and reasonable **proximity queries**
  - this is because **all the words** must be present in a block in order for that block to hold the phrase or the proximity query. Hence the **OR** of all the query masks is searched
- Remark:
  - no other patterns (e.g. range queries) can be searched in this scheme

Phrase/Proximity Queries and **Block Boudaries**

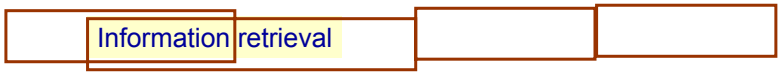
q=<information retrieval>

Text blocks

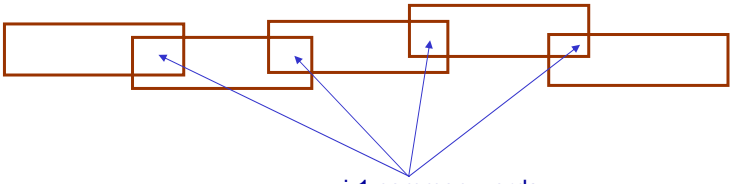


(πρόβλημα! Μπορούμε όμως να το λύσουμε με επικαλυπτόμενα blocks)

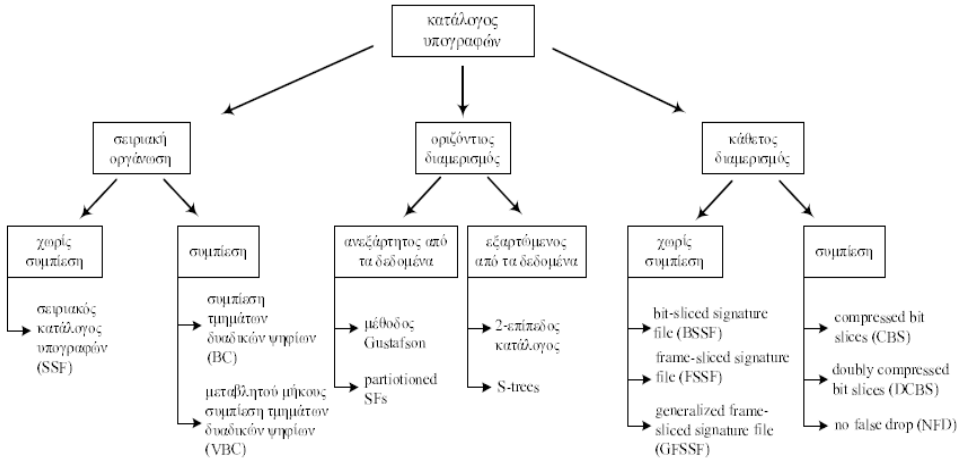
Overlapping blocks



For j-proximity queries



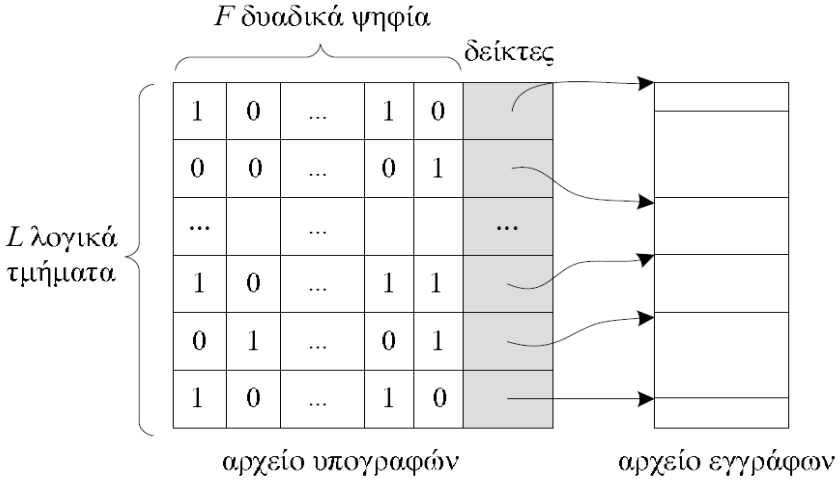
Οργάνωση Υπογραφών



Σειριακή Οργάνωση Υπογραφών

- Η πιο απλή μορφή καταλόγου βασίζεται στη σειριακή παράθεση των υπογραφών σε ένα αρχείο που καλείται **σειριακό αρχείο υπογραφών** (sequential signature file - SSF).
- Το αρχείο υπογραφών είναι στην ουσία ένας πίνακας *L* x *F* με *L* γραμμές (πλήθος blocks - λογικών τμημάτων) και *F* στήλες (πλήθος δυαδικών ψηφίων ανά υπογραφή).
- Σε κάθε υπογραφή αντιστοιχεί και ένα δείκτης (pointer) που δείχνει στην αρχή του λογικού τμήματος του εγγράφου.

Σειριακή Οργάνωση Υπογραφών



Οργάνωση Υπογραφών

- Με βάση τον τρόπο λειτουργίας του καταλόγου SSF προκύπτει ότι για την αναζήτηση ενός και μόνο όρου θα πρέπει να εξεταστούν όλες οι υπογραφές των λογικών τμημάτων.

Εναλλακτικές μορφές οργάνωσης του αρχείου υπογραφών.

Κάθετος Διαμερισμός: BSSF

Τεμαχισμός (slicing) του πίνακα υπογραφών (1988): **BSSF** (bit-sliced signature file).

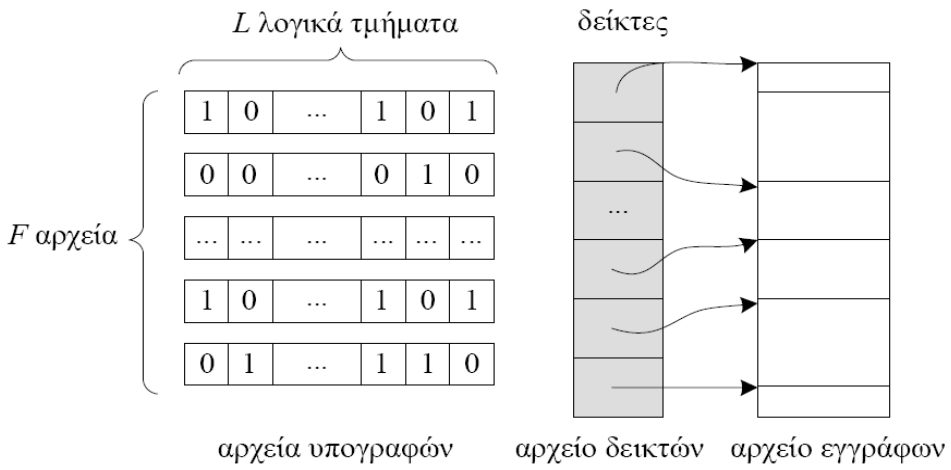
Κάθετος διαμερισμός του πίνακα υπογραφών: η αποθήκευση του πίνακα γίνεται κατά στήλες (και όχι κατά γραμμές όπως στη μέθοδο SSF).
Δηλαδή, ανά bit της υπογραφής

Ο πίνακας υπογραφών του αντιστρέφεται, και αποκτά διαστάσεις F x L (F γραμμές και L στήλες):

> Η κάθε γραμμή του αντεστραμμένου πίνακα καλείται **τεμάχιο** (slice) και αποτελείται από τα δυαδικά ψηφία που βρίσκονται στην ίδια θέση σε όλες τις υπογραφές των λογικών τμημάτων.

Για να μπορεί η δομή να υποστηρίξει εισαγωγές και διαγραφές αποδοτικά, η κάθε γραμμή του αντεστραμμένου πίνακα αποθηκεύεται σε ξεχωριστό αρχείο.

Κάθετος Διαμερισμός: BSSF

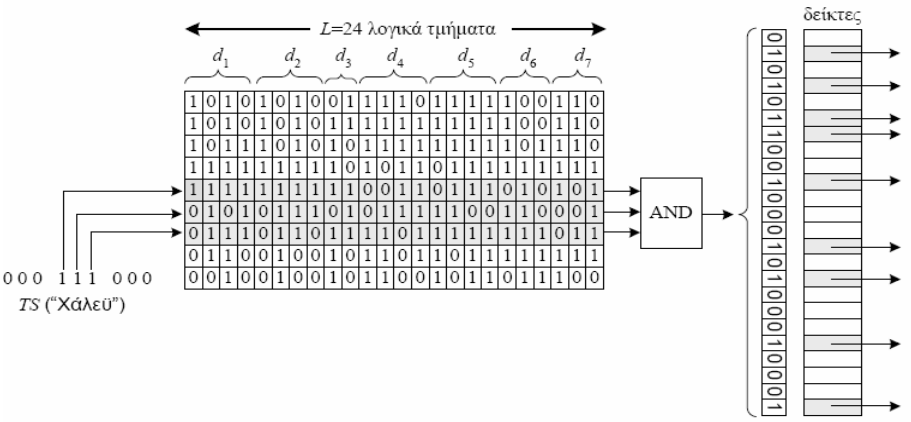


Κάθετος Διαμερισμός: BSSF

Η **αναζήτηση** ενός όρου στη δομή BSSF:

- Υπολογισμός της υπογραφής του όρου. Η υπογραφή του όρου θα περιέχει άσσους σε ακριβώς *m* δυαδικά ψηφία.
- (σε αντίθεση με τη δομή SSF) Εξέταση *m* τεμαχίων (γραμμών του αντεστραμμένου πίνακα) – αυτών που είναι 1 στην υπογραφή
- Τα δυαδικά ψηφία των *m* γραμμών συνδυάζονται με τη χρήση υπέρθεσης (λογικό AND) και προκύπτει ένα διάνυσμα *L* θέσεων.
- Λαμβάνονται υπόψη οι θέσεις των άσσων στο διάνυσμα αυτό και προσπελαύονται οι αντίστοιχοι δείκτες του αρχείου δεικτών για να οδηγηθούμε τελικά στα λογικά τμήματα των εγγράφων.

Κάθετος Διαμερισμός: BSSF



Κάθετος Διαμερισμός: BSSF

Για την **εισαγωγή** ενός νέου εγγράφου,

- αρχικά προσδιορίζονται τα νέα λογικά τμήματα και οι αντίστοιχες υπογραφές.
- για κάθε νέο λογικό τμήμα πραγματοποιείται τεμαχισμός της υπογραφής του και κάθε ένα από τα *F* διαφορετικά αρχεία λαμβάνει και ένα δυαδικό ψηφίο της υπογραφής που αποθηκεύεται στο τέλος.

Κάθετος Διαμερισμός: BSSF

Η μέθοδος BSSF είναι πιο αποδοτική από την SSF ως προς τη λειτουργία της αναζήτησης.

Ωστόσο, υπάρχει επιπλέον χώρος για βελτίωση που οφείλεται σε δύο κυρίως λόγους:

- Η αναζήτηση ενός όρου επιβάλλει την προσπέλαση *m* τεμαχίων, όπου *m* είναι ο αριθμός των άσσων στην υπογραφή του όρου. Αν *m*=1 τότε θα μπορούσε να αυξηθεί η απόδοση της μεθόδου.
- Η εισαγωγή ενός νέου λογικού τμήματος απαιτεί ένα μεγάλο αριθμό προσπελάσεων που ρυθμίζεται από τον αριθμό των δυαδικών ψηφίων της υπογραφής του λογικού τμήματος *F*. Αν η τιμή της παραμέτρου *F* είναι μεγάλη (π.χ. 1000) τότε αυξάνεται σημαντικά το κόστος εισαγωγής.

Σύνοψη

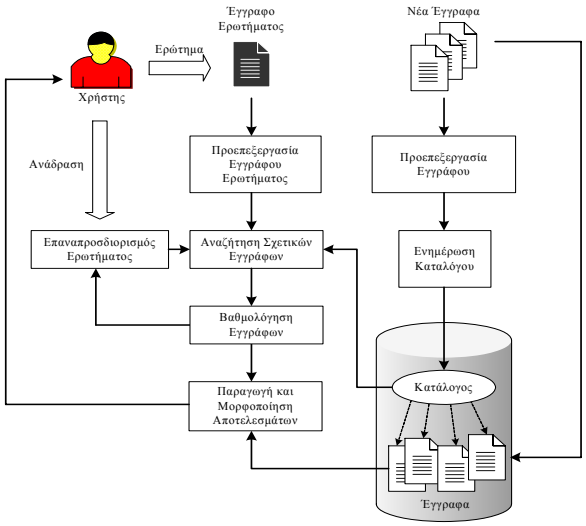
- Οι κατάλογοι υπογραφών αποτελούν μία διαφορετική προσέγγιση για την οργάνωση μίας συλλογής εγγράφων.
- Το βασικό χαρακτηριστικό των καταλόγων αυτών είναι ότι στηρίζονται στη δημιουργία υπογραφών από τους όρους των εγγράφων.
- Μία υπογραφή είναι μία ακολουθία δυαδικών ψηφίων (bits) τα οποία περιέχουν άσσους σε συγκεκριμένες θέσεις που καθορίζονται από τη συνάρτηση κατακερματισμού που χρησιμοποιείται.

Σύνοψη

- Σύμφωνα με πειραματικές μελέτες σχετικά με την επίδοση των καταλόγων υπογραφών σε σχέση με τους ανεστραμμένους καταλόγους, έχει επαληθευτεί ότι οι κατάλογοι που στηρίζονται στην αντιστροφή έχουν γενικά καλύτερες επιδόσεις από τους καταλόγους που στηρίζονται σε υπογραφές.

- Ωστόσο, οι κατάλογοι υπογραφών έχουν μερικές πολύ καλές ιδιότητες (π.χ., ευκολία στον παραλληλισμό)

Δομή ενός ΣΑΠ



Δομές Ευρετηρίου: Διάρθρωση Διάλεξης

- Εισαγωγή – κίνητρο
- Ανεστραμμένα Αρχεία (Inverted files)
- Αρχεία Υπογραφών (Signature files)
- Δένδρα Καταλήξεων (Suffix trees)

Δένδρα και Πίνακες Καταλήξεων
(Suffix Trees and Suffix Arrays)

Δένδρα και Πίνακες Καταλήξεων (Suffix Trees and Arrays)

Κίνητρο

- Γρήγορη αποτίμηση των phrase queries
- Η έννοια της **λέξης** (στην οποία βασίζονται τα inverted files) δεν υπάρχει σε άλλες εφαρμογές (π.χ. στις γενετικές βάσεις δεδομένων), άρα υπάρχει ανάγκη για διαφορετικές δομές δεδομένων.

Μια αλυσίδα DNA είναι μια ακολουθία από διατεταγμένα ζευγάρια βάσεων.
Υπάρχουν 4 βάσεις: η αδενίνη (**A**), η γουανίνη (**G**), η κυτοσίνη (**C**) και η θυμίνη (**T**).
Κάθε ζευγάρι βάσεων του DNA αποτελείται από διαφορετικές βάσεις.
Συγκεκριμένα, η αδενίνη (A) μπορεί να συνδέεται μόνο με τη θυμίνη (T), ενώ η γουανίνη (G) μπορεί να συνδέεται μόνο με την κυτοσίνη (C). Ένα παράδειγμα αποσπάσματος αλυσίδας DNA ακολουθεί:

A G G C T A C C C T T A
T C C G A T G G G A A T

Δένδρα Καταλήξεων (Suffix Trees)

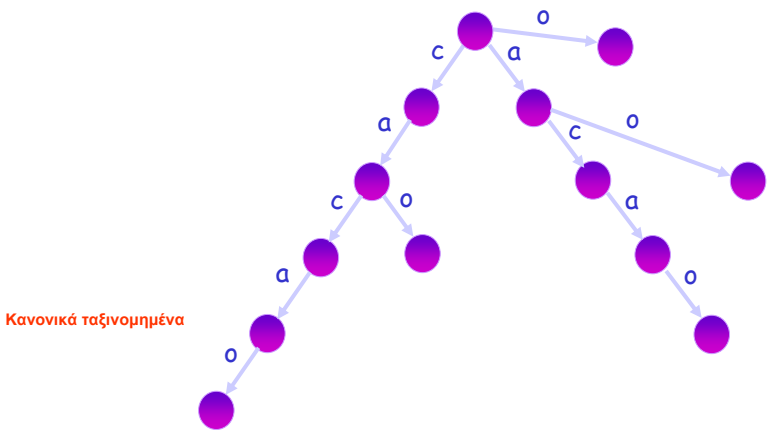
Δένδρο Καταλήξεων:

- Το **δένδρο καταλήξεων** ενός κειμένου είναι ένα **trie** πάνω σε όλες τις καταλήξεις του κειμένου.
- Οι δείκτες προς το κείμενο αποθηκεύονται στα φύλλα του δένδρου.

Suffix Trie για τη λέξη ”cacao”
(θεωρώντας κάθε θέση ως σημείο ευρετηρίου)

Καταλήξεις:
o
ao
cao
acao
cacao

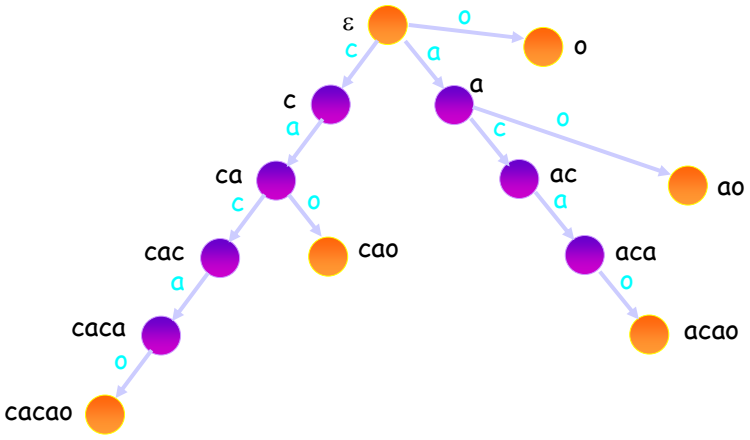
Trie Καταλήξεων



Κανονικά ταξινομημένα

Suffix Trie για τη λέξη ”cacao”
(θεωρώντας κάθε θέση ως σημείο ευρετηρίου)

Καταλήξεις:
o
ao
cao
acao
cacao



Δένδρα και Πίνακες Καταλήξεων (Suffix Trees and Arrays)

Γενική ιδέα

- Βλέπουμε όλο το κείμενο ως μία μακριά συμβολοσειρά (long string)
- Θεωρούμε κάθε θέση του κειμένου ως **κατάληξη κειμένου (text suffix)**
- Δύο καταλήξεις που ξεκινούν από διαφορετικές θέσεις είναι λεξικογραφικά διαφορετικές
  - άρα κάθε κατάληξη προσδιορίζεται μοναδικά από τη θέση της αρχής της

Επιλογές

- Ευρετηριάζουμε όλες τις θέσεις του κειμένου
- Ευρετηριάζουμε κάποιες θέσεις του κειμένου (π.χ. μόνο τις αρχές λέξεων)
  - Άρα εδώ έχουμε την έννοια του σημείου ευρετηρίου (index point)
  - Τα σημεία που δεν είναι σημεία ευρετηρίου δεν είναι παραδόσιμα (deliverable)

Παράδειγμα καταλήξεων

(θεωρώντας ως σημεία ευρετηρίου (index points) τις αρχές των λέξεων)

This is a text. A text has many words. Words are made from letters.

letters.

made from letters.

Words are made from letters.

words. Words are made from letters.

many words. Words are made from letters.

text has many words. Words are made from letters.

text. A text has many words. Words are made from letters.

Δένδρα Καταλήξεων (Suffix Trees)

Δένδρο Καταλήξεων:

- Το **δένδρο καταλήξεων** ενός κειμένου είναι ένα **trie** πάνω σε όλες τις καταλήξεις του κειμένου.
- Οι δείκτες προς το κείμενο αποθηκεύονται στα φύλλα του δένδρου.

Για μείωση του χώρου, το trie συμπυκνώνεται ως ένα Patricia tree

- Patricia = Practical Algorithm To Retrieve Information Coded in Alphanumerical

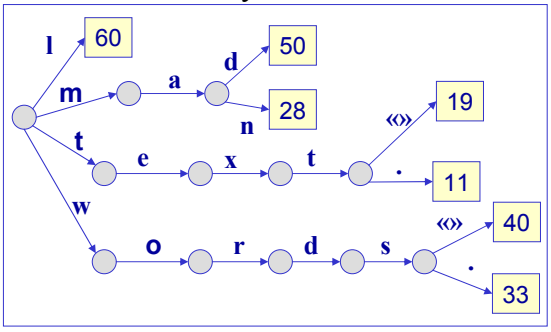
Παράδειγμα καταλήξεων και του αντίστοιχου Suffix Trie

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.

letters.
made from letters.
Words are made from letters.
words. Words are made from letters.
many words. Words are made from letters.
text has many words. Words are made from letters.
text. A text has many words. Words are made from letters.

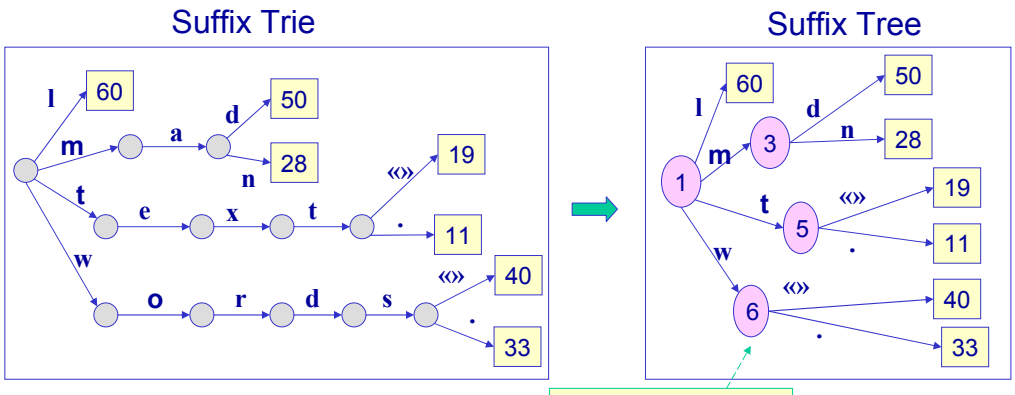
Suffix Trie



Suffix tree
= Suffix trie compacted into a Patricia tree

This involves compressing unary paths, ι.e. paths where each node has just one child.

If unary paths are not present, the tree has O(n) nodes instead of the worst-case $O(n^2)$ of the trie.



Πίνακες Καταλήξεων (Suffix arrays)

Πίνακες Καταλήξεων (Suffix arrays)
(Space efficient implementation of suffix trees)

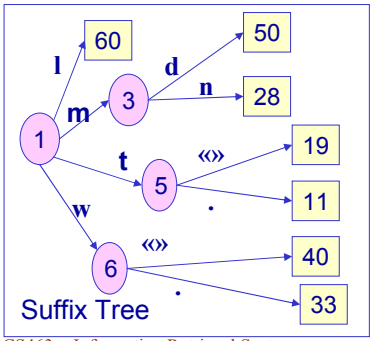
- Suffix trees have a space overhead of 120%-240% over the text size (assuming that index points = word beginnings)
  - assuming node size of 12 or 24 bytes
- Now we will present a data structure with space requirements like those of the inverted file (~40% overhead over the text size)

Πίνακες Καταλήξεων (Suffix arrays)
(Space efficient implementation of suffix trees)

Πίνακας Καταλήξεων:
- Πίνακας με **δείκτες** προς όλες τις «καταλήξεις» σε λεξικογραφική σειρά
- Για να τον δημιουργήσουμε αρκεί μια depth-fist-search διάσχιση του suffix tree.

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.



Suffix Array

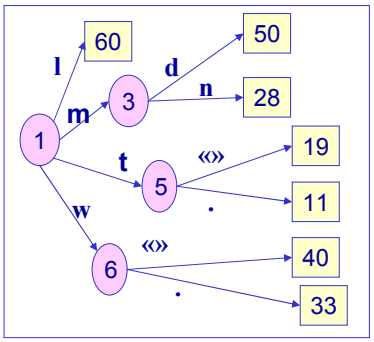
l	m	m	t	t	w	w
60	50	28	19	11	40	33

Πίνακες Καταλήξεων

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.

Suffix Tree



Suffix Array

l	m	m	t	t	w	w

60	50	28	19	11	40	33

Οφέλη:

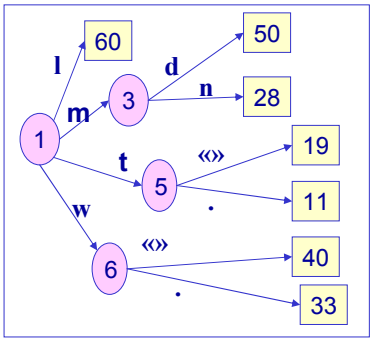
- Μείωση χώρου
  - κρατάμε 1 δείκτη ανά κατάληξη (7 καταλήξεις, πίνακας 7 κελιών)
  - (space overhead ~ that of inverted files)
- Δυνατότητα **binary search**

Πίνακες Καταλήξεων(III)

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.

Suffix Tree



Suffix Array

l	m	m	t	t	w	w

60	50	28	19	11	40	33

Αναζήτηση βάσει Suffix Array

Για να δούμε αν υπάρχει μια κατάληξη στο κείμενο κάνουμε δυαδική αναζήτηση (binary search) στο περιεχόμενο των δεικτών

Πίνακες Καταλήξεων(IV)

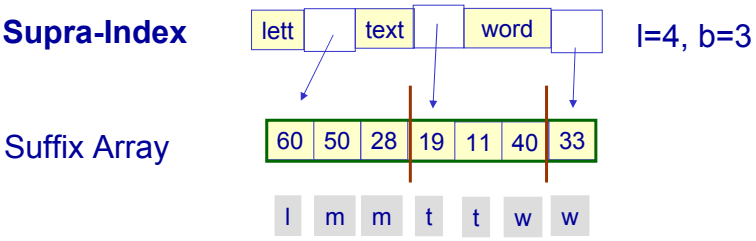
Αναζήτηση βάσει Suffix Array

Για να δούμε αν υπάρχει μια κατάληξη στο κείμενο κάνουμε δυαδική αναζήτηση (binary search) στο περιεχόμενο των δεικτών

Μπορεί να οδηγήσει σε πολλά disk accesses

Therefore if vocabulary is big (and the suffix array does not fit in main memory), **supra indices** are employed

- they store the first l characters for each of every b entries of the suffix array



Πίνακες Καταλήξεων (με supra-index) έναντι Ανεστραμμένων Αρχείων

- For word-indexing suffix array, it has been suggested that a new sample could be taken each time the first word of the suffix changes, and to store the word instead of l characters
- This is exactly as having a vocabulary of the text plus pointers to the array
- The only important difference between this structure and an inverted index is that the occurrences of each word in an inverted index are stored by text position, while in a suffix array they are stored lexicographically by the text following the word.

Κόστος Αποτίμησης Επερωτήσεων

- Κόστος αναζήτησης μιας συμβολοσειράς μήκους m χαρακτήρων
  - O(m) στην περίπτωση των δένδρων καταλήξεων (suffix tree)
  - O(log n) στην περίπτωση των πινάκων καταλήξεων (suffix array)
    - θυμηθείτε οτι κάθε σημείο του κειμένου προσδιορίζει μια κατάληξη
- Αποτίμηση phrase queries
  - Η φράση αναζητείται σαν να ήταν μια συμβολοσειρά
- Αποτίμηση proximity queries
  - proximity queries have to be resolved element wise