Information Retrieval

(A) Προεπεξεργασία Κειμένου
(Text Preprocessing)

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

Yannis Tzitzikas

University of Crete

CS-463,Spring 05

Lecture : 8-9
Date : 17/22-3-2005

Προεπεξεργασία κειμένου : Διάρθρωση Διάλεξης

- Εισαγωγή
- Lexical Analysis (Λεξιλογική ανάλυση)
- Stopwords (Λέξεις Αποκλεισμού)
- Stemming (Στελέχωση Κειμένου)
  - Manual
  - Table Lookup
  - Successor Variety
  - n-Grams
  - Affix Removal (Porter's algorithm)

Προεπεξεργασία Κειμένου

- Κίνητρο
  - δεν είναι όλες οι λέξεις ενός κειμένου κατάλληλες για την παράσταση του περιεχομένου του (μερικές λέξεις φέρουν περισσότερο νόημα από άλλες)
- Προεπεξεργασία
  - προσπάθεια ελέγχου (κυρίως μείωσης) του λεξιλογίου
- Στόχοι της προεπεξεργασίας
  - βελτίωση της αποτελεσματικότητας (effectiveness)
  - βελτίωση της αποδοτικότητας (efficiency) της ανάκτησης
  - μείωση του μεγέθους των ευρετηρίων

Φάσεις Προεπεξεργασίας

[α] Λεξιλογική ανάλυση
  - αναγνώριση *αριθμών*, *λέξεων*, *διαχωριστικών*, *σημείων στίξεως*, κλπ

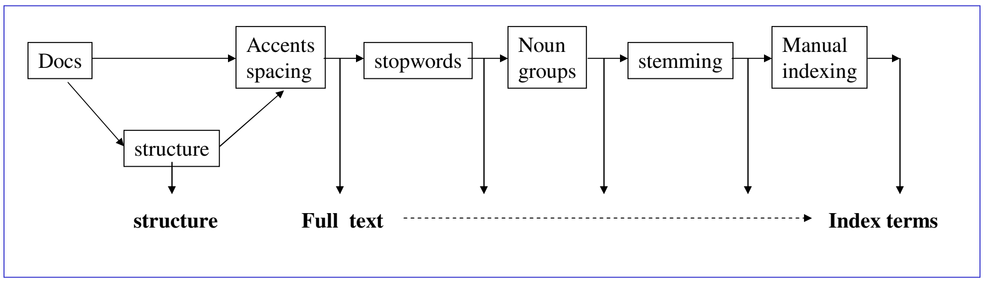
[β] Αποκλεισμός λέξεων (stopwords)
  - απαλοιφή λέξεων με πολύ μικρή διακριτική ικανότητα (*άρθρα*, *αντωνυμίες*, *κτητικές αντωνυμίες*, κλπ)

[γ] Στελέχωση (stemming) των εναπομεινάντων λέξεων
  - απαλοιφή *καταλήξεων/προθεμάτων* (αυτοκίνητο, αυτοκίνητα, αυτοκινήτων) για την ανάκτηση των κειμένων που περιέχουν συντακτικές παραλλαγές των λέξεων της επερώτησης

[δ] Επιλογή των λέξεων που θα χρησιμοποιηθούν στην ευρετηρίαση
  - συχνά γίνεται βάσει του μέρους του λόγου (ουσιαστικά, επίθετα, επιρρήματα, ρήματα)

Φάσεις Προεπεξεργασίας (II)

Από το πλήρες κείμενο στους όρους ευρετηρίου



[α] **Λεξιλογική Ανάλυση** (Lexical Analysis)

Σκοπός: identify tokens

- αναγνώριση *αριθμών*, *λέξεων*, *διαχωριστικών*, *σημείων στίξεως*, κλπ

Περιπτώσεις που απαιτούν προσοχή:

- Λέξεις που περιέχουν ψηφία
  - O2, βιταμίνη B6, B12
- Παύλες (hyphens)
  - “state of the art” vs “state-of-the-art”
  - “Jean-Luc Hainaut”, “Jean-Roch Meurisse”, F-16, MS-DOS
- Σημεία στίξεως (punctuations)
  - OS/2, .NET, command.com
- Μικρά-κεφαλαία
  - συνήθως όλα μετατρέπονται σε μικρά

[α] Λεξιλογική Ανάλυση (ΙΙ)

- Λεξιλογική Ανάλυση για Επερωτήσεις
  - Όπως και για το κείμενο, συν <u>αναγνώριση χαρακτήρων ελέγχου</u>
    - AND, OR, NOT, proximity operators, regular expressions, etc
- Τρόποι υλοποίησης ενός Λεξιλογικού Αναλυτή
  - (α) use a <u>lexical analyzer generator</u> (like lex)
    - best choice if there are complex cases
  - (b) write a lexical analyzer by hand ad hoc,
    - worse choice (error prone)
  - (c) write a lexical analyzer by hand as a <u>finite state machine</u>

[β] **Stopwords** (Λέξεις Αποκλεισμού)

Απαλοιφή λέξεων με πολύ <u>μικρή διακριτική ικανότητα</u> (*άρθρα*, *αντωνυμίες*, *κτητικές αντωνυμίες*, κλπ)

  - e.g. “a”, “the”, “in”, “to”; pronouns: “I”, “he”, “she”, “it”.

- Οφέλη
  - μείωση μεγέθους ευρετηρίου (έως και 40%)
- Παρατηρήσεις
  - Οι λέξεις αποκλεισμού εξαρτώνται από τη γλώσσα και τη συλλογή
  - Not every frequent english word should be in the list
    - Top 200 English words include «time, war, home, life, water, world»
    - In a CS corpus we could add to the stoplist the words: «computer, program, source, machine, language»
- Προβλήματα
  - q=“**to be or not to be**”
  - (για το λόγο αυτό μερικές Μηχανές Αναζήτησης του Ιστού δεν κάνουν προεπεξεργασία)

[β] Stopwords

Τρόποι Υλοποίησης

- 1/ Examine lexical analyzer output and remove stopwords
  - Μπορούμε να αποθηκεύσουμε τις λέξεις αυτές σε έναν hashtable για να τις αναγνωρίζουμε γρήγορα (σε σταθερό χρόνο)
- 2/ Remove stopwords as part of lexical analysis
  - Πιο γρήγορη προσέγγιση αφού η λεξιλογική ανάλυση θα γίνει έτσι και αλλιώς και η αφαίρεση των λέξεων αποκλεισμού δεν απαιτεί επιπλέον χρόνο

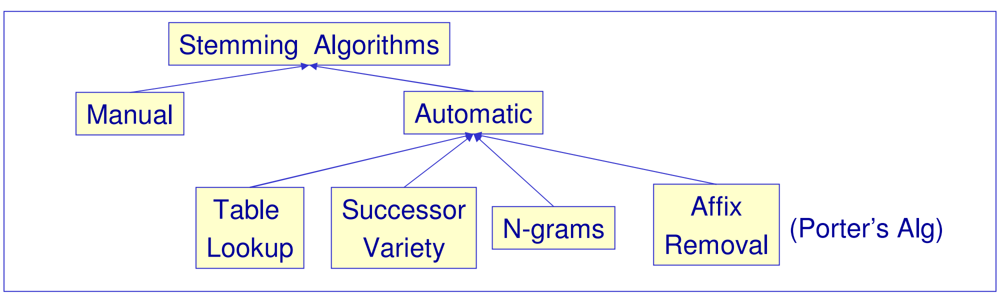
[β] Stopwords: Examples

- ***English**:*
  - *a be had it only she was about because has its of some we after been have last on such were all but he more one than when also by her most or that which an can his mr other the who any co if mrs out their will and corp in ms over there with are could inc mz s they would as for into no so this up at from is not says to*
- French:
- *a afin ah ai aie aient aies ailleurs ainsi ait alentour alias allais allaient allait allons allez alors Ap. Apr. aprs aprs demain arrire as assez attendu au aucun aucune au dedans au dehors au dela au dessous au dessus au devant audit aujourd aujourdhui auparavant auprs auquel aura aurai auraient aurais aurait auras aurez auriez aurions aurons auront aussi aussitôt autant autour autre autrefois autres autrui aux auxdites auxdits auxquelles auxquels avaient avais avait avant avant hier avec avez aviez avions avoir avons ayant ayez ayons B bah banco be beaucoup ben bien bientôt bis bon C c Ca ça ça cahin caha car ce ce ceans ceci cela celle celle ci celle la celles celles ci celles la celui celui ci celui la cent cents cependant certain certaine certaines certains certes ces cest a dire cet cette ceux ceux ci ceux la cf. cg cgr chacun chacune chaque cher chez ci ci ci aprs ci dessous ci dessus cinq cinquante cinquante cinq cinquante deux cinquante et un cinquante huit cinquante neuf cinquante quatre cinquante sept cinquante six cinquante trois cl cm cm combien comme comment contrario contre crescendo D d dabord daccord daffilee dailleurs dans daprs darrache pied davantage de debout dedans dehors deja dela demain demblee depuis derechef derrire des ds desdites desdits desormais desquelles desquels dessous dessus deux devant devers dg die differentes differents dire dis disent dit dito divers diverses dix dix huit dix neuf dix sept dl dm donc dont dorenavant douze du dû dudit duquel durant E eh elle elle elles elles en en en encore enfin ensemble ensuite entre entre temps envers environ es s est et et/ou etaient etais etait etant etc ete êtes etiez etions être eu eue eues euh eûmes eurent eus eusse eussent eusses eussiez eussions eut eût eûtes eux exprs extenso extremis F facto fallait faire fais faisais faisait faisaient faisons fait faites faudrait faut fi flac fors fort forte fortiori frais fûmes fur furent fus fusse fussent fusses fussiez fussions fut fût fûtes G GHz gr grosso gure H ha han haut he hein hem heu hg hier hl hm hm hola hop hormis hors hui huit hum I ibidem ici ici bas idem il il illico ils ils ipso item J j jadis jamais je je jusqu jusqua jusquau jusquaux jusque juste K kg km km^2 L l la la la la la bas la dedans la dehors la derrire la dessous la dessus la devant la haut laquelle lautre le le lequel les les ls lesquelles lesquels leur leur leurs lez loin lon longtemps lors lorsqu lorsque lui lui lun lune M m m m ma maint mainte maintenant maintes maints mais mal malgre me même mêmes mes mg mgr MHz mieux mil mille milliards millions minima ml mm mm^2 modo moi moi moins mon moult moyennant mt N n nagure ne neanmoins neuf ni nº non nonante nonobstant nos notre nous nous nul nulle O ô octante oh on on ont onze or ou où ouais oui outre P par parbleu parce par ci par dela par derrire par dessous par dessus par devant parfois par la parmi partout pas passe passim pendant personne petto peu peut peuvent peux peut être pis plus plusieurs plutôt point posteriori pour pourquoi pourtant prealable prs presqu presque primo priori prou pu puis puisqu puisque Q qu qua quand quarante quarante cinq quarante deux quarante et un quarante huit quarante neuf quarante quatre quarante sept quarante six quarante trois quasi quatorze quatre quatre vingt quatre vingt cinq quatre vingt deux quatre vingt dix quatre vingt dix huit quatre vingt dix neuf quatre vingt dix sept quatre vingt douze quatre vingt huit quatre vingt neuf quatre vingt onze quatre vingt quatorze quatre vingt quatre quatre vingt quinze quatre vingts quatre vingt seize quatre vingt sept quatre vingt six quatre vingt treize quatre vingt trois quatre vingt un quatre vingt une que quel quelle quelles quelqu quelque quelquefois quelques quelques unes quelques uns quelquun quelquune quels qui quiconque quinze quoi quoiqu quoique R revoici revoila rien S s sa sans sauf se secundo seize selon sensu sept septante sera serai seraient serais serait seras serez seriez serions serons seront ses si sic sine sinon sitôt situ six soi soient sois soit soixante soixante cinq soixante deux soixante dix soixante dix huit soixante dix neuf soixante dix sept soixante douze soixante et onze soixante et un soixante et une soixante huit soixante neuf soixante quatorze soixante quatre soixante quinze soixante seize soixante sept soixante six soixante treize soixante trois sommes son sont soudain sous souvent soyez soyons stricto suis sur sur le champ surtout sus T t t ta tacatac tant tantôt tard te tel telle telles tels ter tes toi toi ton tôt toujours tous tout toute toutefois toutes treize trente trente cinq trente deux trente et un trente huit trente neuf trente quatre trente sept trente six trente trois trs trois trop tu tu U un une unes uns USD V va vais vas vers veut veux via vice versa vingt vingt cinq vingt deux vingt huit vingt neuf vingt quatre vingt sept vingt six vingt trois vis a vis vite vitro vivo voici voila voire volontiers vos votre vous vous W X y y Z zero*

[γ] **Stemming (Στελέχωση Κειμένου)**

- Υποβίβαση λέξεων στη ρίζα τους για ανεξαρτησία από τις μορφολογικές παραλλαγές των λέξεων
  - «αυτοκίνητο», «αυτοκίνητα», «αυτοκινήτων»
  - “computer”, “computational”, “computation” all reduced to same token “compute”
- Στόχοι
  - Βελτίωση αποτελεσματικότητας
  - Μείωση του μεγέθους του ευρετηρίου

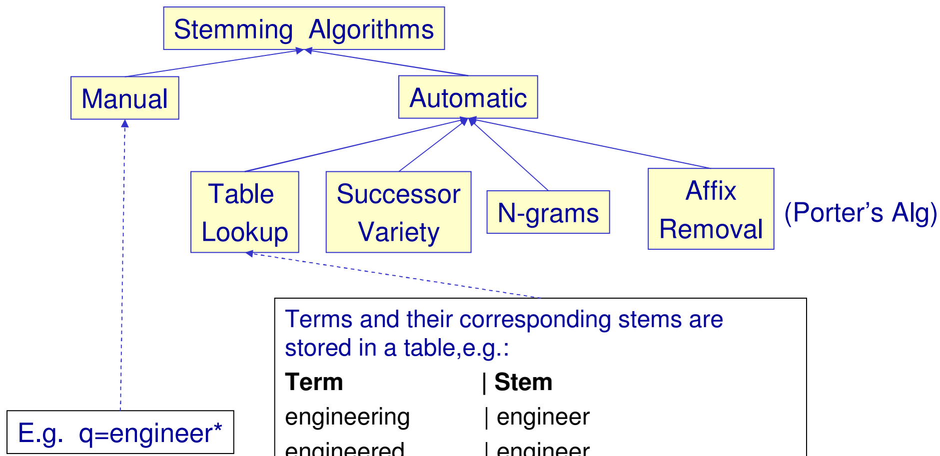
[γ] Stemming Algorithms



How we evaluate a stemming algorithm ?

- Correctness
  - overstemming vs understemming
- Retrieval effectiveness
- Compression performance

[γ] Stemming Algorithms (II)



Terms and their corresponding stems are stored in a table,e.g.:

Term	**Stem**
engineering	engineer
engineered	engineer
engineer	engineer

(such tables are not easily available)

Stemming Algorithms: Successor Variety

- Idea: Use the frequencies of letter sequences in a body of text as the basis for stemming.
- Παράδειγμα
  - Word: READABLE
  - Corpus: ABLE, APE, BEATABLE, FIXABLE, READ, READABLE, READING, READS, RED, ROPE, RIPE

Prefix	**Successor Variety**	**Letters**
R	3	E,I,O
RE	2	A,D
REA	1	D
READ	3	A,I,S
READA	1	B
READAB	1	L
READABL	1	E
READABLE	1	*BLANK*

Stemming Algorithms: Successor Variety (II)

Βήματα για Στελέχωση Κειμένου

1/ Δημιουργία του πίνακα Ποικιλίας Διαδόχων (successor variety table)

2/ Χρήση του πίνακα για <u>τεμαχισμό</u> των λέξεων

π.χ. READABLE => READ ABLE

3/ <u>Επιλογή</u> ενός τεμαχίου (as stem)

π.χ. READABLE => <u>READ</u> ABLE

- **Τεμαχισμός** βάσει peak and plateau method:
  - πχ τεμαχισμός στο γράμμα που οι διάδοχοί του είναι **περισσότεροι** των διαδόχων του προηγούμενου γράμματος
    - REA (1), READ (3)
- **Επιλογή** Τεμαχίου
  - if (first segment occurs in <=12 words in corpus) select first segment, else the second
  - Motivation: If occurs >12 then it is probably a prefix
- Συμπέρασμα
  - Η τεχνική αυτή δεν απαιτεί καμία είσοδο από το σχεδιαστή

Stemming Algorithms: n-grams

Ιδέα: Ομαδοποίησε λέξεις βάσει του αριθμού των κοινών διγραμμάτων ή ν-γραμμάτων

Πχ: σύγκριση "**statistics**" με "**statistical**"

- "statistics":
  - digrams: st ta at ti is st ti ic cs (9)
  - unique digrams: at cs ic is st ta ti (7)
- "statistical":
  - digrams: st ta at ti is st ti ic ca al (10)
  - unique digrams: al at ca ic is st ta ti (8)
- Έχουν 6 κοινά digrams. Dice similarity = 2*6/(7+8)=0.8

- Οι όροι ομαδοποιούνται με αυτόν τον τρόπο (όλες οι λέξεις που έχουν την ίδια ρίζα καταχωρούνται στην ίδια ομάδα)

Stemming Algorithms: Affix Removal

- Idea: Remove Suffixes and/or Prefixes

- Instance: **Porter's Stemmer**
  - Simple procedure for removing known affixes in English without using a dictionary.
  - Can produce unusual stems that are not English words:
    - "computer", "computational", "computation" all reduced to same token "comput"
  - May conflate (reduce to the same token) words that are actually distinct.
  - Not recognize all morphological derivations.

Stemming Algorithms: Porter Stemmer

- Παραδείγματα κανόνων:
  - s → ∅ (for plural form)
  - sses → ∅ (for plural form)
- Εφαρμόζεται πρώτα η μακρύτερη ακολουθία
  - e.g. stresses => stress, NOT stresses => stresse

Stemming Algorithms: Porter Stemmer > Rules

suffix	replacement	example
1a		
sses	ss	caresses->caress
ies	i	ponies->poni, ties->tie
ss	NUL	cats->cat
1b		
eed	ee	agreed->agree
ed	NUL	plastered->plaster
ing	NUL	motoring->motor
2		
ational	ate	relational->relate
tional	tion	conditional->condition
izer	ize	digitizer->digitize
ator	ate	operator->operate
….		

Stemming Algorithms: Porter Stemmer >Errors

- Errors of “comission”:
  - organization, organ → organ
  - police, policy → polic
  - arm, army → arm
- Errors of “omission”:
  - cylinder, cylindrical
  - create, creation
  - Europe, European

Stemming Algorithms: Porter Stemmer > Code

- Δείτε [MIR, Appendix]

- Demo available at:
  - http://snowball.tartarus.org/demo.php

- Implementation (C, Java, …) available at:
  - http://www.tartarus.org/~martin/PorterStemmer/

[δ] Επιλογή Λέξεων για την Ευρετηρίαση

- Πχ μόνο ουσιαστικά,
- Ελεγχόμενα λεξιλόγια - Θησαυροί, ...

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

Δομές Ευρετηρίου: Διάρθρωση Διάλεξης

- Εισαγωγή - κίνητρο
- Inverted files (ανεστραμμένα αρχεία)
- Suffix trees (δένδρα καταλήξεων)
- Signature files (αρχεία υπογραφών)
- Sequential Text Searching
- Answering Pattern-Matching Queries

Ευρετηρίαση Κειμένου:Εισαγωγή

- Κίνητρο
  - **Δομές 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**
- Εδώ
  - **σχεδιασμός δομών δεδομένων, που ονομάζονται ευρετήρια (called *indices*), για επιτάχυνση της αναζήτησης**

Ανάγκες Γλωσσών Επερώτησης

- 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
  - Boolean queries
  - phrase/proximity queries
  - pattern matching
  - Regular expressions
  - structured text
  - ...

Τεχνικές Ευρετηρίασης (Indexing Techniques)

- Inverted files (ανεστραμένα αρχεία)
  - η πιο διαδομένη τεχνική
- Suffix trees and arrays (δένδρα και πίνακες καταλήξεων)
  - γρήγορες για phrase queries αλλά η κατασκευή και η συντήρησή τους είναι δυσκολότερη
- Signature files (αρχεία υπογραφών)
  - δημοφιλείς τη δεκαετία του 80 αλλά σήμερα τα ανεστραμμένα αρχεία υπερτερούν

Υπόβαθρο/Επανάληψη: **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)

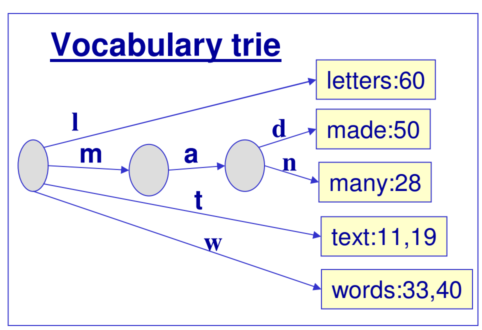
Tries: Παράδειγμα

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)



Inverted Files (Ανεστραμένα αρχεία)

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

- Δομή:
  - Vocabulary: is the set of all distinct words in the text
  - Occurrences: lists containing all information necessary for each word of the vocabulary (text position, frequency, documents where the word appears, etc.)

Παράδειγμα

Κείμενο

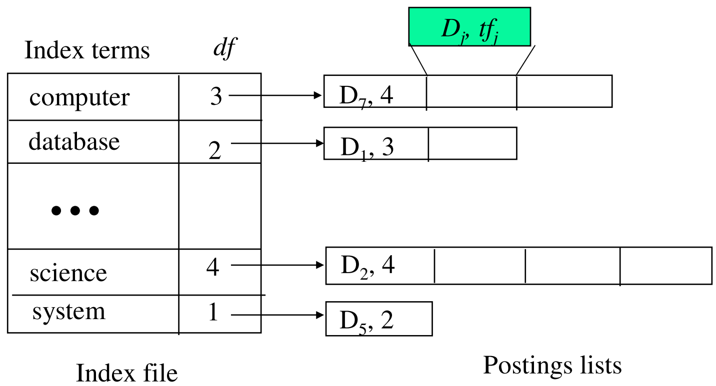
That house has a garden. The garden has many flowers. The flowers are beautiful

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

Inverted Files for the Vector Space Model



Απαιτήσεις Χώρου Space Requirements

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

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

Notations
- *n*: the size of the text
- *m*: the length of the pattern (m << n)
- v: the size of the vocabulary
- *M*: the amount of main memory available

Block Addressing

- The text is divided in blocks
- The occurrences point to the blocks where the word appears
- 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 are obtained with this technique)
- Disadvantages:
  - online sequential search over the qualifying blocks if exact positions are required

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**

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

Index	Small collection (1Mb)		Medium collection (200Mb)		Large collection (2Gb)	
Addressing words	45%	73%	36%	64%	35%	63%
Addressing documents	19%	26%	18%	32%	26%	47%
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

reduction

Searching an inverted index

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 occurences)

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**
- The structures most used to store the vocabulary are ***hashing, tries*** or ***B-trees***
  - *cost of hashing: O(m)*
  - *cost of tries: O(m)*
- An alternative is simply storing the words in **lexicographical order**
  - cheaper in space and very competitive
  - cost of binary search: *O(log V)*

1/ Vocabulary Search (II)

- Remarks
  - prefix and range queries can also be solved with binary search, tries or B-trees buts 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 synchorization to find places where all the words appear in sequence (for a phrase) or appear close enough (for proximity)
  - 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.

Inverted Index: Κατασκευή

- 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
  - this can be done efficiently using a trie data structure
- If it is not found, it is added to the vocabulary with a empty list of occurrences and the new position is added to the end of its list of occurrences

Inverted Index: Κατασκευή (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. This allows the vocabulary to be kept in memory at search time
- The overall process is *O(n)* worst-case time

Trie:

O(1) per text character

Since positions are appended O(1) time

Overall process O(n)

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

A technique based on **partial Indexes:**

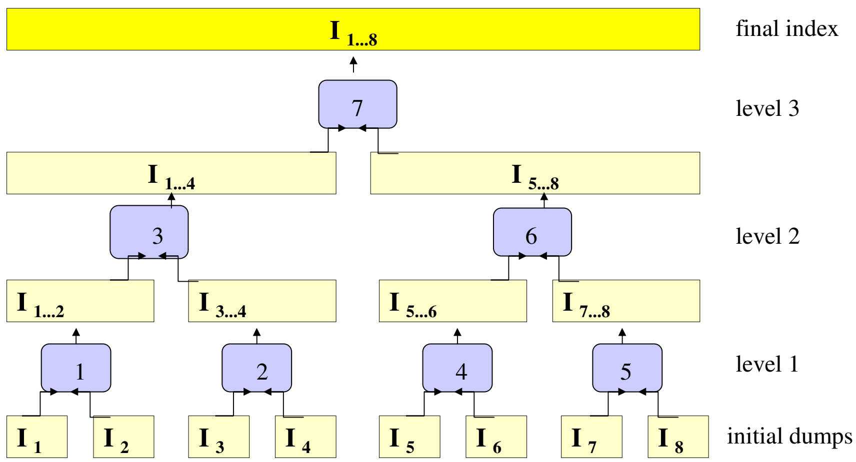
- Use the previous algorithm until the main memory is exhausted.
- When no more memory is available, **write to disk** the **partial index I_i** obtained up to now, and **erase it from main memory**
- Continue with the rest of the text

- 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 ot the larger-numbered index, the therefore the lists are just **concatenated**
- Complexity: O(n1+n2) where n1 and n2 the sizes of the indices

Merging partial indices to obtain the final



Merging all partial indices: Complexity

- 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 final index

- Addition of a new doc
  - build its index and merge it 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))

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

Suffix Trees and Arrays (Δένδρα και Πίνακες Καταλήξεων)

- 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)
- Γενική ιδέα
  - Βλέπουμε όλο το κείμενο ως ένα μακρύ string
  - Θεωρούμε κάθε θέση του κειμένου ως **κατάληξη κειμένου (text suffix)**
  - 2 καταλήξεις που εκκινούν από διαφορετικές θέσεις είναι λεξικογραφικά διαφορετικές
    - άρα κάθε κατάληξη προσδιορίζεται μοναδικά από τη θέση της αρχής του
  - Δεν είναι υποχρεωτικό να ευρετηριάσουμε όλες τις θέσεις του κειμένου
    - Index points = beginnings (e.g. word beginnings)
    - the elements which are not beginnings are not deliverable

Suffix Trees and Arrays (II)

- Μειονεκτήματα
  - Η κατασκευή τους είναι ακριβή
  - Τα αποτελέσματα (του ψαξίματος) δεν διανέμονται με βάση τη σειρά εμφάνισής τους στο κείμενο
  - Καταλαμβάνουν πολύ χώρο
    - even if only word beginnings are indexed, we have a space overhead of 120% to 240%

Παράδειγμα καταλήξεων

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 Trees

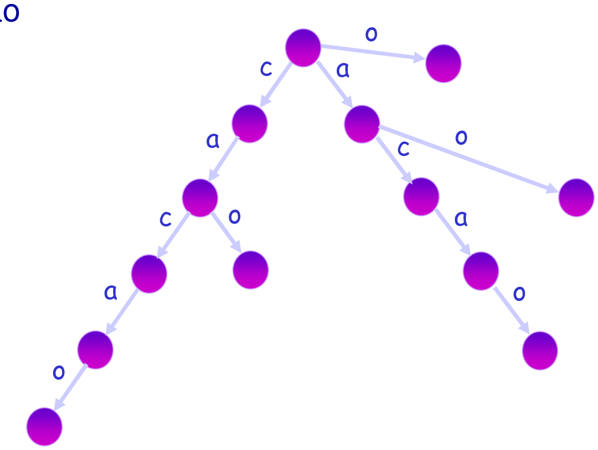
Ορισμός

- **Suffix tree** = **trie** built over **all the suffixes** of the text
- Οι δείκτες αποθηκεύονται στα φύλλα
- Για μείωση του χώρου, το trie συμπυκνώνεται ως ένα Patricia tree
  - Patricia = Practical Algorithm To Retrieve Information Coded in Alphanumerical

Suffix Trie για τη λέξη ”cacao”

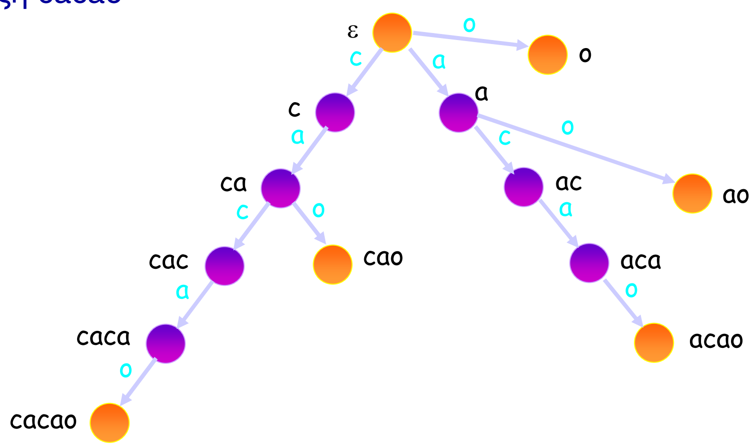
Για τη λέξη cacao

Suffixes:
o
ao
cao
acao
cacao

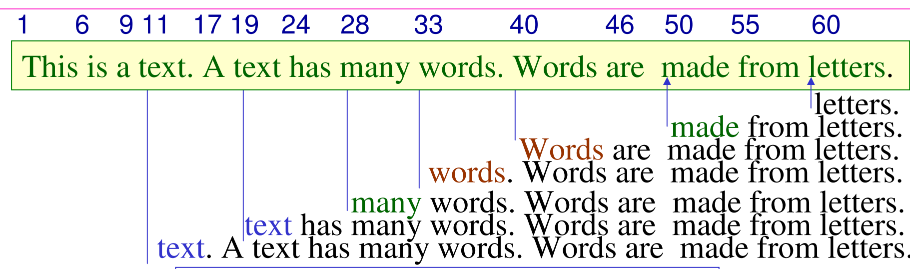


Για τη λέξη cacao

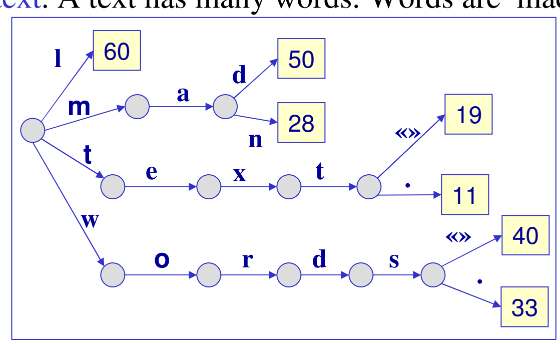
Suffixes:
- o
- ao
- cao
- acao
- cacao



Suffix Trie

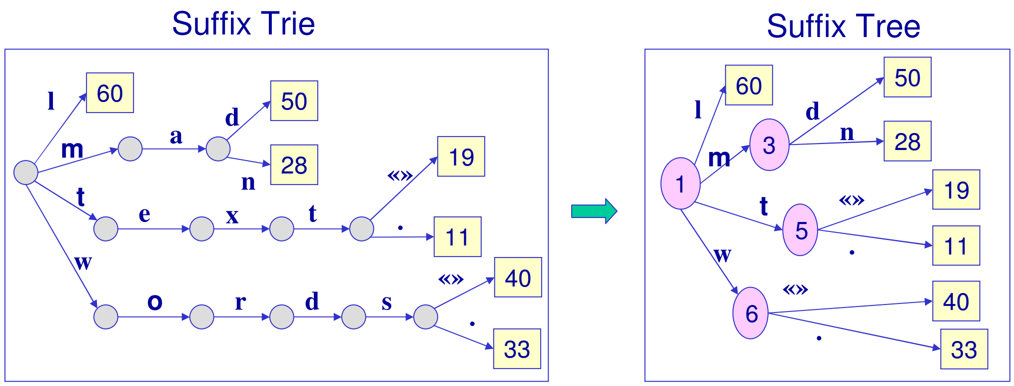


Suffix Trie



Suffix tree
= Suffix trie compacted into a Patricia tree

– this involves compressins unary paths, ι.e. paths where each node has just one child.
– Once unary paths are not present, the tree has O(n) nodes instead of the worst-case O(n^2) of the trie



Suffix arrays
(Space efficient implementation of suffix trees)

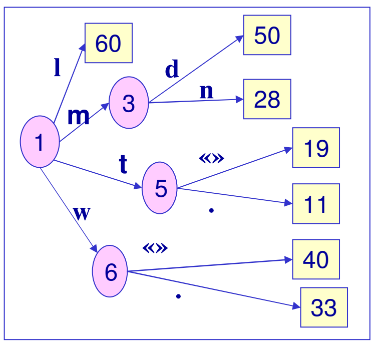
- Suffix Array:
  – Πίνακας με όλες τις καταλήξεις σε λεξικογραφική σειρά
  – Για να τον δημιουργήσουμε αρκεί μια depth-fist-search διάσχιση του suffix tree
- Οφέλη
  – Μείωση χώρου
    - 1 δείκτη ανά κατάληξη (overhead ~ that of inverted files)
  – Δυνατότητα **binary search**

Suffix arrays (II)

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

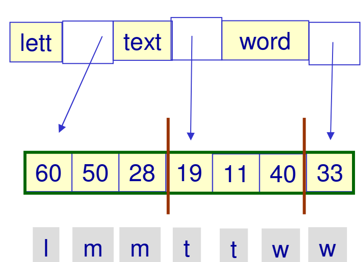
l	m	m	t	t	w	w
60	50	28	19	11	40	33

Suffix Arrays (II)

- 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

Supra-Index

Suffix Array



l=3, b=3

Searching

- Evaluating phrase queries
  - Η φράση αναζητείται ωσάν να ήταν ένα string **(...)**
    - proximity queries have to be resolved element wise
- Cost of searching a string of m characters
  - O(m) in case of suffix tree
  - O(log n) in case of suffix array

Δομές Ευρετηρίου: Διάρθρωση Διάλεξης

- Εισαγωγή - κίνητρο
- Inverted files (ανεστραμμένα αρχεία)
- Suffix trees (δένδρα καταλήξεων)
- **Signature files (αρχεία υπογραφών)**
- Sequential Text Searching
- Answering Pattern-Matching Queries

Signature Files (Αρχεία Υπογραφών)

Κύρια σημεία:

- Δομή ευρετηρίου που βασίζεται στο **hashing**
- Μικρή χωρική επιβάρυνση (**10%-20%** του μεγέθους του κειμένου)
- Αναζήτηση = σειριακή αναζήτηση στο αρχείο υπογραφών
- Κατάλληλη για όχι πολύ μεγάλα κείμενα

Συγκεκριμένα

- Χρήση hash function που αντιστοιχεί λέξεις κειμένου σε bit masks των B bits
- **Διαμέριση** του κειμένου σε blocks των b λέξεων το καθένα
- Bit mask of a block = **Bitwise OR** of the bits masks of all words in the block
- Bit masks are then concatenated

Αρχεία Υπογραφών: Παράδειγμα

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

	Block 1	Block 2	Block 3	Block 4
Text	This is a text.	A text has many	words. Words are	made from letters.
Text Signature	000101	110101	100100	101101

Signature Function

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

Αρχεία Υπογραφών: Αναζήτηση

Έστω ότι αναζητούμε μια λέξη w:

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

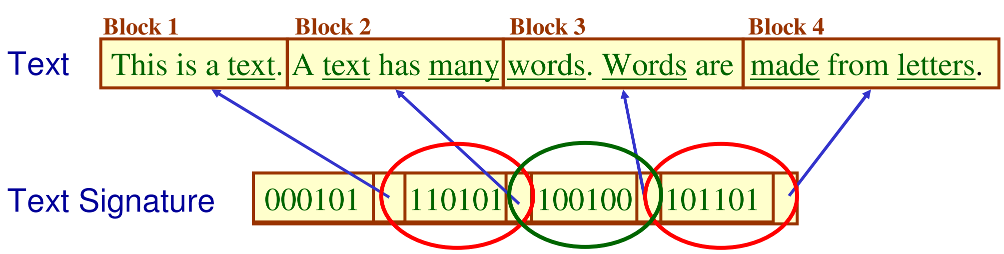
2/ Compare W with all bit masks Bi of all text blocks
If (W **&** Bi = W), the text block i is candidate (may contain the word w)

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: All bits of the W are set in Bi but the word w is not there

w=«words», h(«words»)=100100



Signature Function

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

Configuration (Διαμόρφωση)

- 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=log2(V)
- Παράμετροι:
  - B (το μέγεθος των bit mask)
  - L (L<B) to 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

Άλλες Παρατηρήσεις

- Μέγεθος αρχείου υπογραφών:
  - bit masks of each block plus one pointer for each block
- Συντήρηση αρχείου υπογραφών:
  - Η προσθήκη/διαγραφή αρχείων αντιμετωπίζεται εύκολα
    - προσθέτονται/διαγράφονται τα αντίστοιχα bit masks

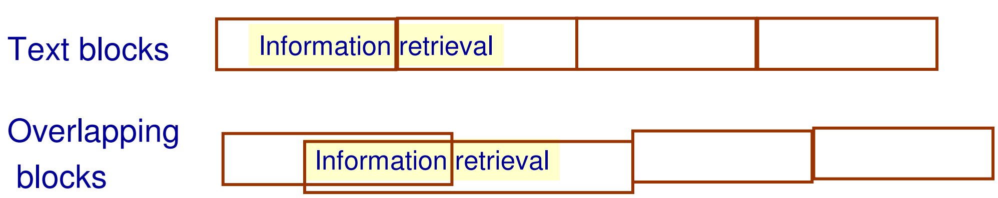
Signature files: Phrase and Proximity Queries

- 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

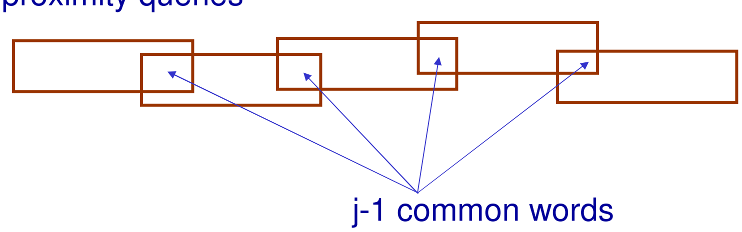
- Remark:
  - no other patterns (e.g. range queries) can be searched in this scheme

Phrase/Proximity Queries and **Block Boudaries**

q=<information retrieval>



For j-proximity queries



Δομές Ευρετηρίου: Διάρθρωση Διάλεξης

- Εισαγωγή - κίνητρο
- Inverted files (ανεστραμμένα αρχεία)
- Suffix trees (δένδρα καταλήξεων)
- Signature files (αρχεία υπογραφών)
- **Sequential Text Searching**
- Answering Pattern-Matching Queries

Sequential Text Searching

- Brute-Force Algorithm
- Knuth-Morris-Pratt
- Boyer-Moore family

Αναζήτηση Κειμένου: Το πρόβλημα

find the first occurrence (or all occurrences) of a string (or pattern) *p* (of length *m*) in a string *s* (of length *n*)

Commonly, *n* is much larger than *m*.

Brute-Force Algorithm

- *Brute-Force* (BF), or *sequential* text searching:
  - **Try all** possible positions in the text. For each position verify whether the pattern matches at that position.
- Since there are *O(n)* text positions and each one is examined at *O(m)* worst-case cost, the worst-case of brute-force searching is *O(nm)*.

Brute-Force Algorithm

```
Naive-String-Matcher(S,P)
n ← length(S)
m ← length(P)
for i ← 0 to n-m do
        if P[1..m] = Σ[i+1 .. i+m] then
                return "Pattern occurs at position i"
        fi
od
```

The naive string matcher needs worst case running time $O((n-m+1)\ m)$

For $n = 2m$ this is $O(n^2)$

Its average case is $O(n)$ (since on random text a mismatch is found after $O(1)$ comparisons on average)

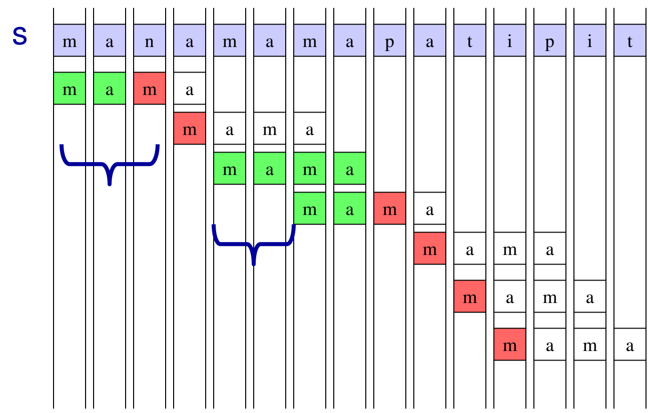
The naive string matcher is not optimal, since string matching can be done in time $O(m + n)$

Knuth-Morris-Pratt & Boyer-Moore

- Πιο γρήγοροι αλγόριθμοι που βασίζονται **μετακινούμενο παράθυρο**
- Γενική ιδέα:
  - They employ a ***window*** of length *m* which is slid over the text.
  - It is *checked* whether the text in the window is equal to the pattern (if it is, the window position is reported as a match).
  - Then, the window is *shifted* forward.
- Οι αλγόριθμοι διαφέρουν στον τρόπο που ελέγχουν και μετακινούν το παράθυρο.

Η γενική ιδέα

p="mama"



- It does not try all window positions as BF does. Instead, it reuses information from previous checks.

Knuth-Morris-Pratt (KMP) [1970]

- The pattern *p* is preprocessed to build a table called *next*.
- The *next* table at position *j* says which is the longest proper prefix of *p[1..j-1]* which is also a suffix and the characters following prefix and suffix are different.
- Hence *j-next[j]-1* window positions can be safely skipped if the characters up to *j-1* matched and the *j*-th did not.

KMP: the next table

next[j] = longest proper prefix of *p[1..j-1]* which is also a suffix and the characters following prefix and suffix are different

j	1	2	3	4	5	6	7	8	9	10	11	
p[j]	a	b	r	a	c	a	d	a	b	r	a	
next[j]	0	0	0	0	1	0	1	0	0	0	0	4

KMP: the next table

next[j] = longest proper prefix of *p[1..j-1]* which is also a suffix and the characters following prefix and suffix are different

j	1	2	3	4	5	6	7	8	9	10	11	
p[j]	a	b	r	a	c	a	d	a	b	r	a	
next[j]	0	0	0	0	1	0	1	0	0	0	0	4

KMP: the next table

next[j] = longest proper prefix of *p[1..j-1]* which is also a suffix and the characters following prefix and suffix are different

j	1	2	3	4	5	6	7	8	9	10	11	
p[j]	a	b	r	a	c	a	d	a	b	r	a	
next[j]	0	0	0	0	1	0	1	0	0	0	0	4

KMP: the next table

next[j] = longest proper prefix of *p[1..j-1]* which is also a suffix and the characters following prefix and suffix are different

j	1	2	3	4	5	6	7	8	9	10	11	
p[j]	a	b	r	a	c	a	d	a	b	r	a	
next[j]	0	0	0	0	1	0	1	0	0	0	0	4

KMP: the next table

next[j] = longest proper prefix of *p[1..j-1]* which is also a suffix and the characters following prefix and suffix are different

j	1	2	3	4	5	6	7	8	9	10	11	
p[j]	a	b	r	a	c	a	d	a	b	r	a	
next[j]	0	0	0	0	1	0	1	0	0	0	0	4

KMP: the next table

next[j] = longest proper prefix of *p[1..j-1]* which is also a suffix and the characters following prefix and suffix are different

j	1	2	3	4	5	6	7	8	9	10	11	
p[j]	a	b	r	a	c	a	d	a	b	r	a	
next[j]	0	0	0	0	1	0	1	0	0	0	0	4

KMP: the next table

next[j] = longest proper prefix of *p[1..j-1]* which is also a suffix and the characters following prefix and suffix are different

j	1	2	3	4	5	6	7	8	9	10	11	
p[j]	a	b	r	a	c	a	d	a	b	r	a	
next[j]	0	0	0	0	1	0	1	0	0	0	0	4

KMP: the next table

next[j] = longest proper prefix of *p[1..j-1]* which is also a suffix and the characters following prefix and suffix are different

j	1	2	3	4	5	6	7	8	9	10	11	
p[j]	a	b	r	a	c	a	d	a	b	r	a	
next[j]	0	0	0	0	1	0	1	0	0	0	0	4

KMP: the next table

next[j] = longest proper prefix of *p[1..j-1]* which is also a suffix and the characters following prefix and suffix are different

j	1	2	3	4	5	6	7	8	9	10	11	
p[j]	a	b	r	a	c	a	d	a	b	r	a	
next[j]	0	0	0	0	1	0	1	0	0	0	0	4

KMP: the next table

next[j] = longest proper prefix of *p[1..j-1]* which is also a suffix and the characters following prefix and suffix are different

j	1	2	3	4	5	6	7	8	9	10	11	
p[j]	a	b	r	a	c	a	d	a	b	r	a	
next[j]	0	0	0	0	1	0	1	0	0	0	0	4

KMP: the next table

next[j] = longest proper prefix of *p[1..j-1]* which is also a suffix and the characters following prefix and suffix are different

j	1	2	3	4	5	6	7	8	9	10	11	
p[j]	a	b	r	a	c	a	d	a	b	r	a	
next[j]	0	0	0	0	1	0	1	0	0	0	0	4

KMP: the next table

next[j] = longest proper prefix of *p[1..j-1]* which is also a suffix and the characters following prefix and suffix are different

j	1	2	3	4	5	6	7	8	9	10	11	
p[j]	a	b	r	a	c	a	d	a	b	r	a	
next[j]	0	0	0	0	1	0	1	0	0	0	0	4

Exploiting the next table

next[j] = longest proper prefix of *p[1..j-1]* which is also
a suffix and the characters following prefix and suffix are different

j	1	2	3	4	5	6	7	8	9	10	11	
p[j]	a	b	r	a	c	a	d	a	b	r	a	
next[j]	0	0	0	0	1	0	1	0	0	0	0	4
j-next[j]-1	0	1	2	3	3	5	5	7	8	9	10	7

- *j-next[j]-1* window positions can be safely skipped if the characters up to *j-1* matched and the *j*-th did not.

Example: match until 2nd char

j	1	2	3	4	5	6	7	8	9	10	11	
p[j]	a	b	r	a	c	a	d	a	b	r	a	
next[j]	0	0	0	0	1	0	1	0	0	0	0	4
j-next[j]-1	0	<u>1</u>	2	3	3	5	5	7	8	9	10	7

s	a	a	r	i	c	a	b	r	a	c	a		
p	a	b	r	a	c	a	d	a	b	r	a		
		a	b	r	a	c	a	d	a	b	r	a	

1

Example: match until 3rd char

j	1	2	3	4	5	6	7	8	9	10	11	
p[j]	a	b	r	a	c	a	d	a	b	r	a	
next[j]	0	0	0	0	1	0	1	0	0	0	0	4
j-next[j]-1	0	1	**2**	3	3	5	5	7	8	9	10	7

s	a	b	a	i	c	a	b	r	a	c	a		
p	a	b	r	a	c	a	d	a	b	r	a		
			a	b	r	a	c	a	d	a	b	r	a

2

Example: match until 7th char

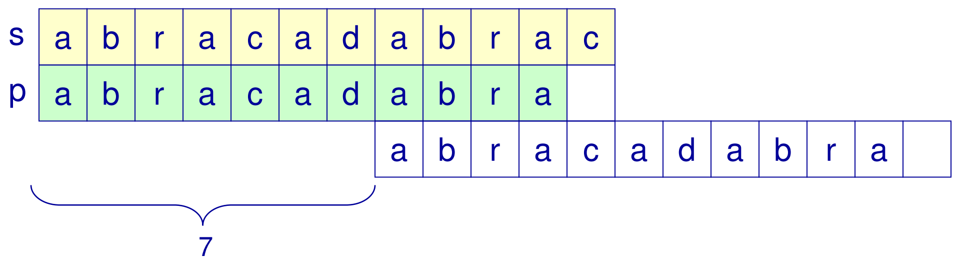
j	1	2	3	4	5	6	7	8	9	10	11	
p[j]	a	b	r	a	c	a	d	a	b	r	a	
next[j]	0	0	0	0	1	0	1	0	0	0	0	4
j-next[j]-1	0	1	2	3	3	5	**5**	7	8	9	10	7

s	a	b	r	a	c	a	b	r	a	c	a					
p	a	b	r	a	c	a	d	a	b	r	a					
						a	b	r	a	c	a	d	a	b	r	a

5

Example: pattern matched

j	1	2	3	4	5	6	7	8	9	10	11	
p[j]	a	b	r	a	c	a	d	a	b	r	a	
next[j]	0	0	0	0	1	0	1	0	0	0	0	4
j-next[j]-1	0	1	2	3	3	5	5	7	8	9	10	<u>7</u>



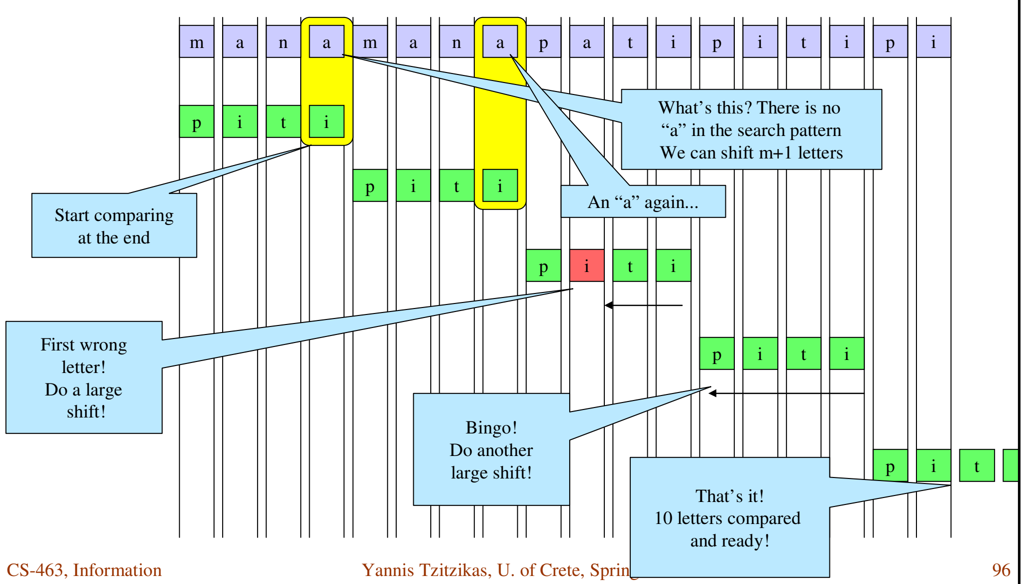
KMP: Complexity

- Since at each text comparison the window or the text pointer advance by at least one position, the algorithm performs at most *2n* comparisons (and at least *n*).
- On average is it not much faster than BF

Boyer-Moore (BM) [1975]

- Motivation
  - KMP yields genuine benefits only if a mismatch as preceded by a partial match of some length
    - only in this case is the pattern slides more than 1 position
  - Unfortunately, this is the exception rather than the rule
    - mathes occur much more seldom than mismatches
- The idea
  - start comparing characters at the **end of the pattern** rather than at the beginning
  - like in KMP, a pattern is pre-processed

Boyer-Moore: The idea by an example



Finite Automata (επανάληψη)

A deterministic finite automaton M is a 5-tuple (Q,q_0,A,Σ,δ), where

- Q is a finite set of **states**
- $q_0 \in Q$ is the **start state**
- $A \subseteq Q$ is a distinguished set of **accepting sates**
- Σ, is a finite **input alphabet**,
- $\delta: Q \times \Sigma \to Q$ is called the **transition function** of M

Let $\varphi : \Sigma \to Q$ be the final-state function defined as:

For the empty string ε we have: $\varphi(\varepsilon) := q_0$

For all $a \in \Sigma$, $w \in \Sigma^*$ define $\varphi(wa) := \delta(\varphi(w), a)$

M accepts w if$_{\text{and only i}}$f: $\varphi(w) \in A$

Example (I)

p=«abba»

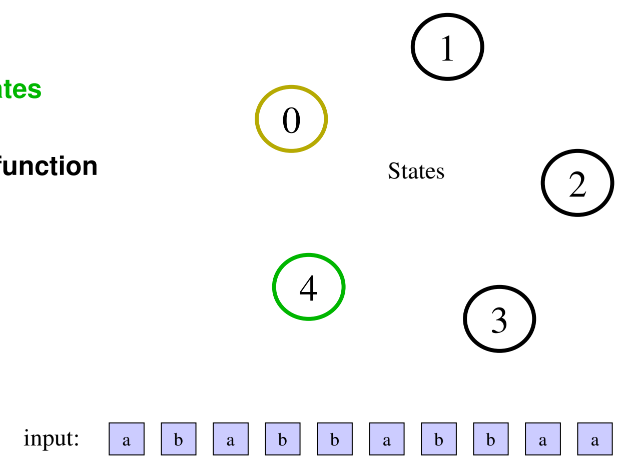
Q is a finite set of states

$q_0 \in Q$ is the start state

Q is a set of accepting sates

Σ: input alphabet

$\delta: Q \times \Sigma \to Q$: transition function



Example (II)

p=«abba»

Q is a finite set of states

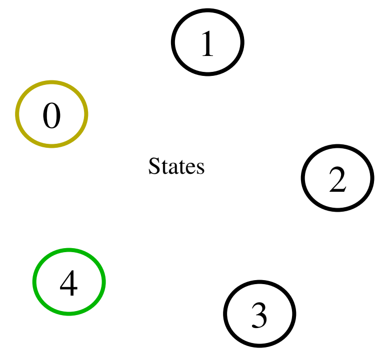
$q_0 \in$ Q is the start state

Q is a set of accepting states

Σ: input alphabet

δ: Q × Σ → Q: transition function

state \ input	a	b
0	1	0
1	1	2
2	1	3
3	4	0
4	1	2



Example (III)

p=«abba»

Q is a finite set of states

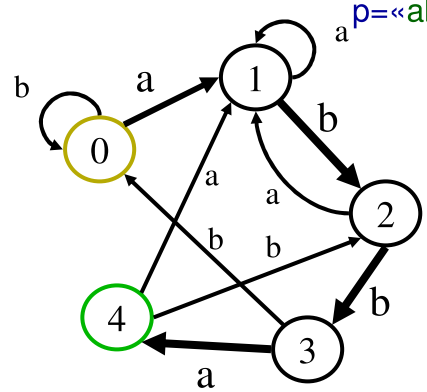
$q_0 \in$ Q is the start state

Q is a set of accepting sates

Σ: input alphabet

δ: Q × Σ → Q: transition function

state \ input	a	b
0	1	0
1	1	2
2	1	3
3	4	0
4	1	2



Example (IV)

Q is a finite set of states

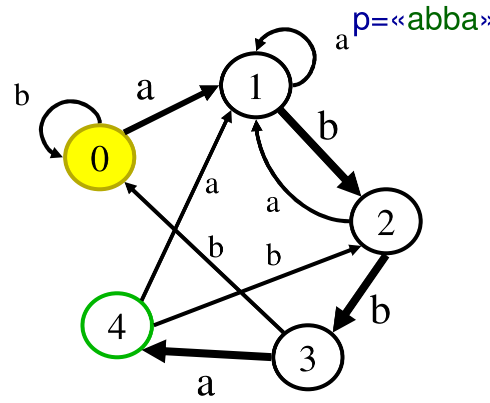
$q_0 \in$ Q is the start state

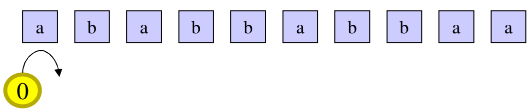
Q is a set of accepting sates

Σ: input alphabet

δ: Q × Σ → Q: transition function

state \ input	a	b
0	1	0
1	1	2
2	1	3
3	4	0
4	1	2





Example (V)

Q is a finite set of states

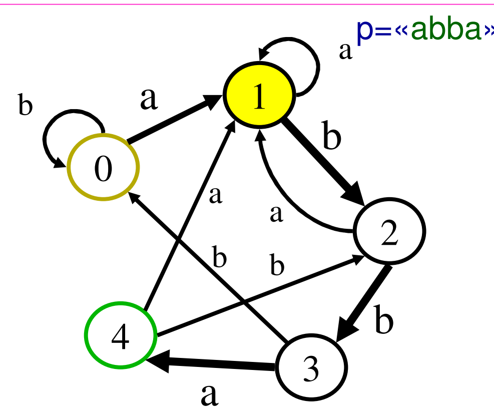
$q_0 \in$ Q is the start state

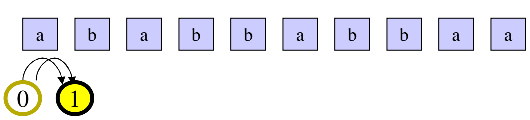
Q is a set of accepting sates

Σ: input alphabet

δ: Q × Σ → Q: transition function

state \ input	a	b
0	1	0
1	1	2
2	1	3
3	4	0
4	1	2





Example (VI)

Q is a finite set of states

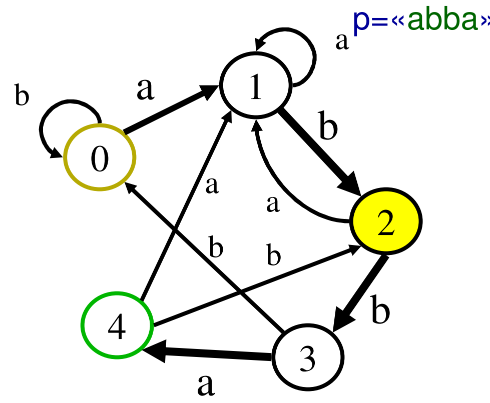
$q_0 \in$ Q is the start state

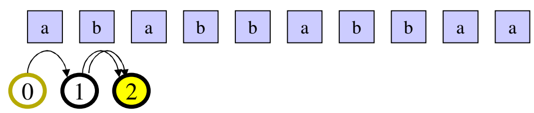
Q is a set of accepting sates

Σ: input alphabet

δ: Q × Σ → Q: transition function

input / state	a	b
0	1	0
1	1	2
2	1	3
3	4	0
4	1	2





Example (VII)

Q is a finite set of states

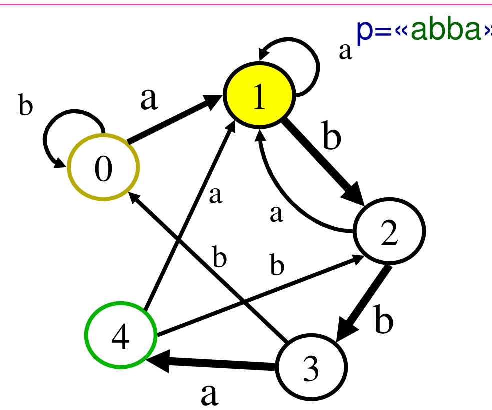
$q_0 \in$ Q is the start state

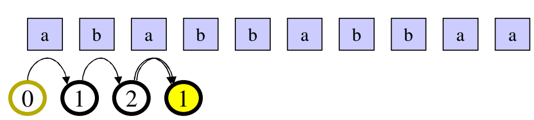
Q is a set of accepting sates

Σ: input alphabet

δ: Q × Σ → Q: transition function

input / state	a	b
0	1	0
1	1	2
2	1	3
3	4	0
4	1	2





Example (VIII)

Q is a finite set of states

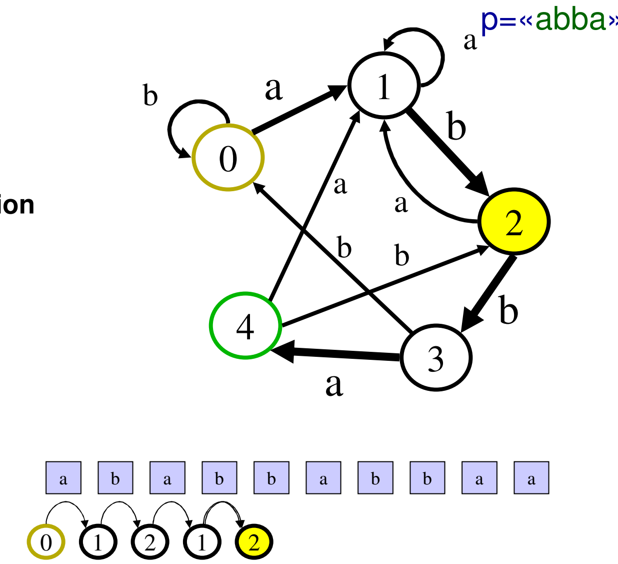
$q_0 \in Q$ is the start state

Q is a set of accepting sates

Σ: input alphabet

δ: Q × Σ → Q: transition function

input / state	a	b
0	1	0
1	1	2
2	1	3
3	4	0
4	1	2



Example (IX)

Q is a finite set of states

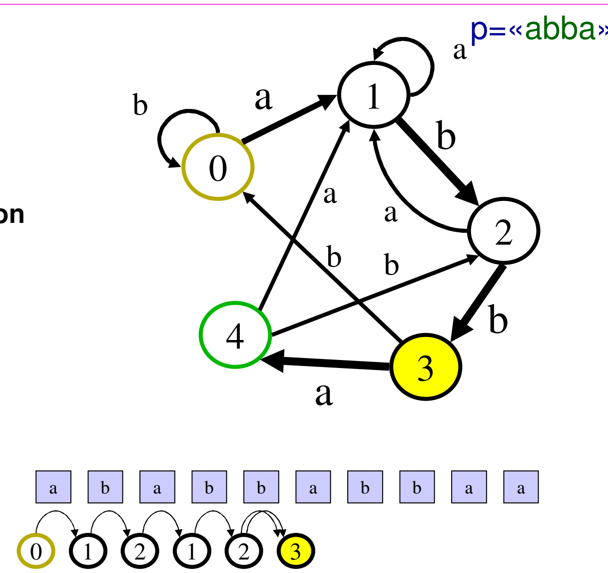
$q_0 \in Q$ is the start state

Q is a set of accepting sates

Σ: input alphabet

δ: Q × Σ → Q: transition function

input / state	a	b
0	1	0
1	1	2
2	1	3
3	4	0
4	1	2



Example (X)

Q is a finite set of states

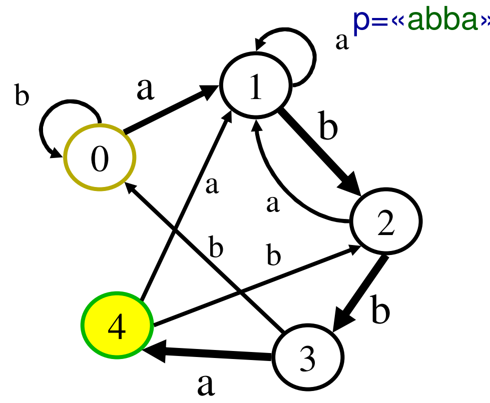
$q_0 \in$ Q is the start state

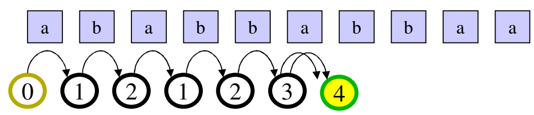
Q is a set of accepting sates

Σ: input alphabet

δ: Q × Σ → Q: transition function

input / state	a	b
0	1	0
1	1	2
2	1	3
3	4	0
4	1	2





Example (XI)

Q is a finite set of states

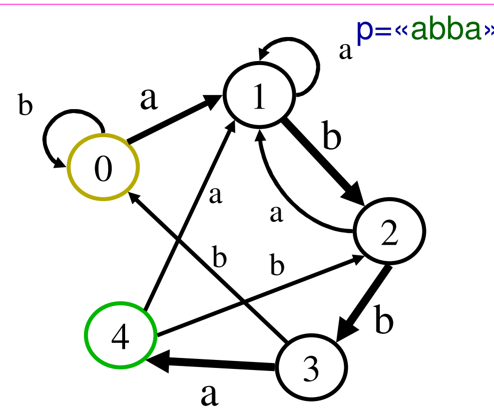
$q_0 \in$ Q is the start state

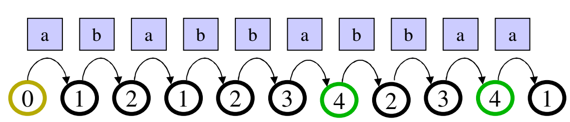
Q is a set of accepting sates

Σ: input alphabet

δ: Q × Σ → Q: transition function

input / state	a	b
0	1	0
1	1	2
2	1	3
3	4	0
4	1	2



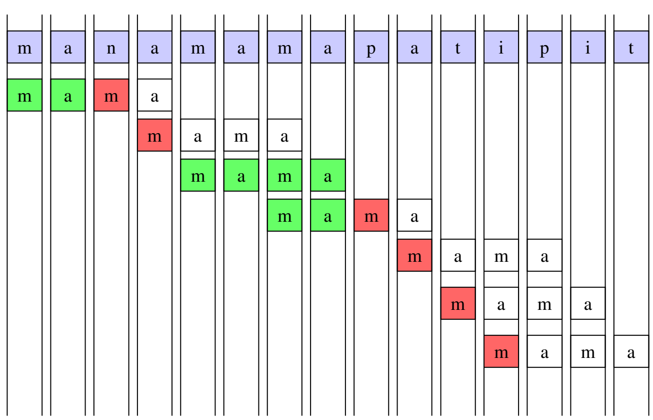


Finite-Automaton-Matcher

- For every pattern of length m there exists an automaton with m+1 states that solves the pattern matching problem with the following algorithm:

```
Finite-Automaton-Matcher(T,δ,P)
n ← length(T)
q ← 0
for i ← 1 to n do
    q ← δ(q,T[i])
    if q = m then
          s ← i - m
          return “Pattern occurs with shift” s
    fi
od
```


Computing the Transition Function: The Idea!



How to Compute the Transition Function?

- **Let P_k denote the first k letter string of P**

```
Compute-Transition-Function(P, Σ)
    m ← length(P)
    for q ← 0 to m do
        for each character a ∈ Σ do
          k ← 1+min(m,q+1)
          repeat
              k ← k-1
          until P_k is a suffix of P_q a
           δ(q,a) ← k
        od
    od
```


Other string searching algorithms

- Shift-Or
- Suffix Automaton
- ...

Δομές Ευρετηρίου: Διάρθρωση Διάλεξης

- Εισαγωγή - κίνητρο
- Inverted files (ανεστραμμένα αρχεία)
- Suffix trees (δένδρα καταλήξεων)
- Signature files (αρχεία υπογραφών)
- Sequential Text Searching
- **Answering Pattern-Matching Queries**

Answering Pattern Matching Queries

- Searching Allowing Errors (Levenshtein distance)
- Searching using Regular Expressions

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**

Remember: 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

Searching Allowing Errors

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

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 (II)**

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

C: m x n matrix // one row for each char of the P, one column for each char of T
C[0,j] = 0 // no letter of P has been consumed
C[i,0] = i // i chars of P have been consumed, pointer of T at 0 (so i errors so far)

C[i,j]=
C[i-1,j-1], **if** P[i]=T[i] // 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 // ~ char 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 + ☐

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 + ☐

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 +

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**

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

Solution using **Dynamic Programming: Example**

- T = “surgery”, P = “survey”, k=**2**

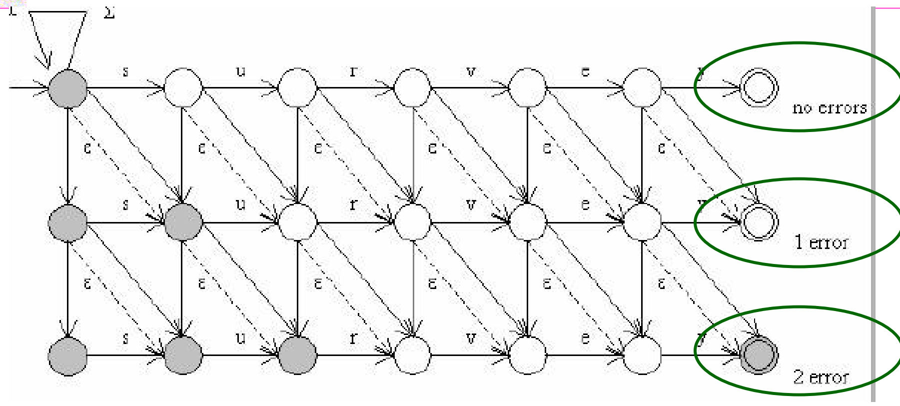
T

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**

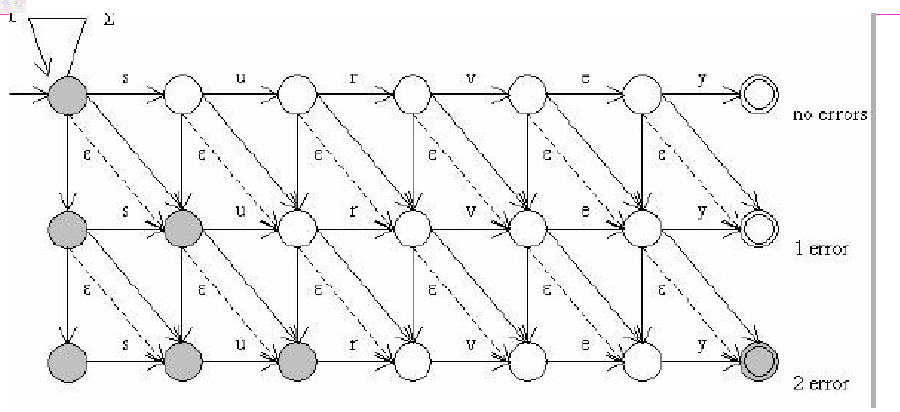
- Cost: O(*mn*) time where *m* and *n* are the lengths of the two strings being compared.
- **O(m) space** as we need to keep <u>only the previous column stored</u>

Searching Allowing Errors: Solution with a Nondeterministic Automaton



- Every column represents matching to pattern up to a given position.

Searching Allowing Errors: Solution with a Nondeterministic Automaton



- At each iteration, a new text character is read and automaton changes its state.
- **Horizontal** arrows represents <u>matching</u> a document.
- **Vertical** arrows represent <u>insertions</u> into pattern
- **Solid diagonal** arrows represent <u>replacements</u>.
- **Dashed diagonal** arrows represent <u>deletion</u> in the pattern (ε: empty).

Searching Allowing Errors: Solution with a Nondeterministic Automaton

- If we convert NDFA into a DFA then it will be huge in size (although the search time will be O(n))
- An alternative solution is **BIT-Parallelism**

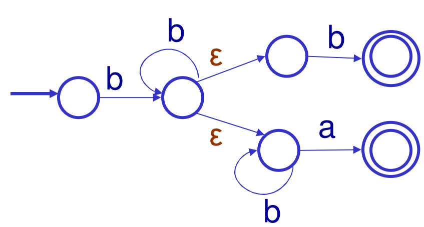
Searching using Regular Expressions

Classical Approach
(a) Build a ND Automaton
(b) Convert this automaton to deterministic form

(a) Build a ND Automaton

Size O(m) where m the size of the regular expression

Π.χ. regex = b b* (b | b* a)

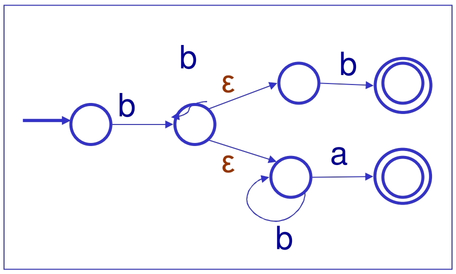


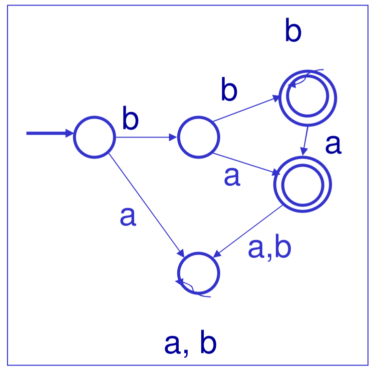
Searching using Regular Expressions (II)

(b) Convert this automaton to deterministic form

- It can search any regular expression in O(n) time where n the size of text
- However, its size and construction time can be exponential in m, i.e. O(m 2^m).

b b* (b | b* a) = (b b* b | b b* b* a) = (b b b* | b b*a)

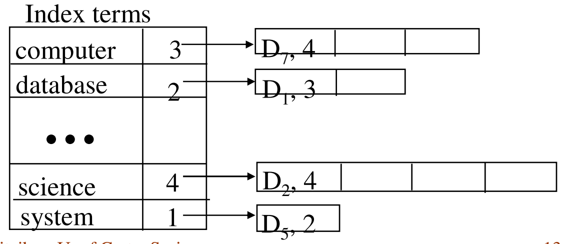




Bit-Parallelism to avoid constructing the deterministic automaton (NFA Simulation)

Pattern Matching Using **Inverted Files**

- Προηγουμένως είδαμε πως μπορούμε να αποτιμήσουμε επερωτήσεις με κριτήρια τύπου Edit Distance, RegExpr, <u>ανατρέχοντας στα κείμενα</u>.
- Τι κάνουμε αν έχουμε ήδη ένα Inverted File ?
  - Ψάχνουμε το Λεξιλόγιο αντί των κειμένων (αρκετά μικρότερο σε μέγεθος)
  - Βρίσκουμε τις λέξεις που ταιριάζουν
  - Συγχωνεύουμε τις λίστες εμφανίσεων (occurrence lists) των λέξεων που ταίριαξαν.



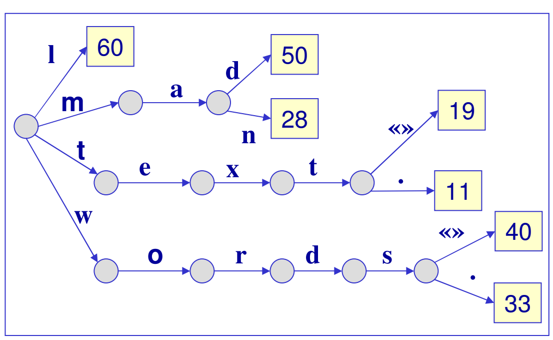
Pattern Matching Using Inverted Files (II)

- If a **block addressing** is used, the search must be completed with a sequential search over the blocks.
- Technique of inverted files is not able to efficiently find approximate matches or regular expressions that span many words.

Pattern Matching Using **Suffix Trees**

- Τι κάνουμε αν έχουμε ήδη ένα Suffix Tree?
- Μπορούμε να αποτιμήσουμε τις επερωτήσεις εκεί, αντί στα κείμενα;

Suffix Trie

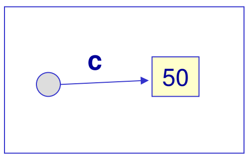


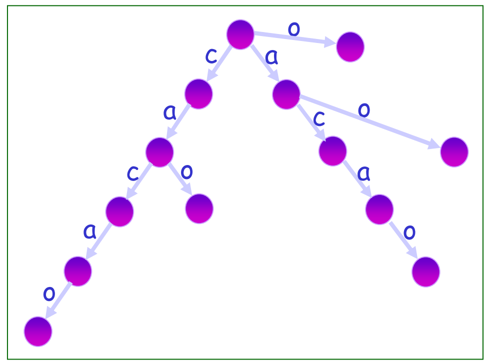
Pattern Matching Using **Suffix Trees**

If the suffix trees index **all text positions** (not just word beginnings) it can search for words, prefixes, suffixes and sub-stings with the same search algorithm and cost described for word search.

Indexing all text positions normally makes the suffix array size 10 times or more the text size.

cacao

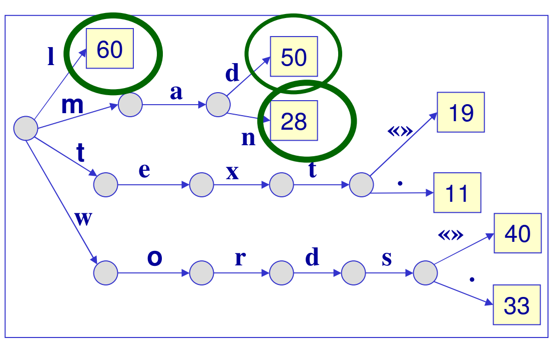




Pattern Matching Using **Suffix Trees (II)**

- **Range queries** are easily solved by just searching both extreme in the trie and then collecting all the leaves lie in the middle.

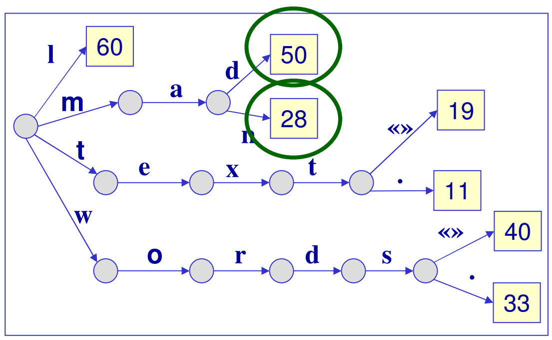
"letter" < q < "many"



Pattern Matching Using **Suffix Trees (II)**

- **Regular expressions** can be searched in the suffix tree. The algorithm simply simulates sequential searching of the regular expression

q=ma*



Δομές Ευρετηρίου: Διάρθρωση Διάλεξης

- Εισαγωγή - κίνητρο
- Inverted files (ανεστραμμένα αρχεία)
- Suffix trees (δένδρα καταλήξεων)
- Signature files (αρχεία υπογραφών)
- Sequential Text Searching
- Answering Pattern-Matching Queries
  - directly on documents
    - Searching Allowing Errors
    - Searching using Regular Expressions
  - on indices (inverted files and suffix trees)


STOP