

# Indexing and Searching

Berlin Chen 2005

## References:

1. Modern Information Retrieval, chapter 8
2. Information Retrieval: Data Structures & Algorithms, chapter 5
3. [G.H. Gonnet, R.A. Baeza-Yates, T. Snider, Lexicographical Indices for Text: Inverted files vs. PAT trees](#)

# Introduction

- **Sequential or online searching**
  - Find the occurrences of a pattern in a text when the **text is not preprocessed**
  - Appropriate when:
    - The text is small
    - Or the text collection is **very volatile**
    - Or the index space overhead cannot be afforded
- **Indexed search**
  - Build data structures over the text (indices) to speed up the search
  - Appropriate for the larger or semi-static text collection
  - The system updated at reasonably regular intervals

# Introduction

- Three **data structures** for indexing are considered

- **Inverted files**

- The best choice for most applications

- **Signature files**

- Popular in the 1980s

- **Suffix arrays**

- Faster but harder to build and maintain

Issues:

Search cost,  
Space overhead,  
Building/updating time

# Inverted Files

- **Basic Ideas**

- A word-oriented mechanism for indexing a text collection in order to speed up the searching task
- Two elements:
  - A vector containing all the distinct words (called vocabulary) in the text collection
  - For each vocabulary word, a list of all docs (**identified by doc number in ascending order**) in which that word occurs

- Distinction between inverted file or list

- **Inverted file**: occurrence points to documents or file names (identities)
- **Inverted list**: occurrence points to word positions

# Inverted Files

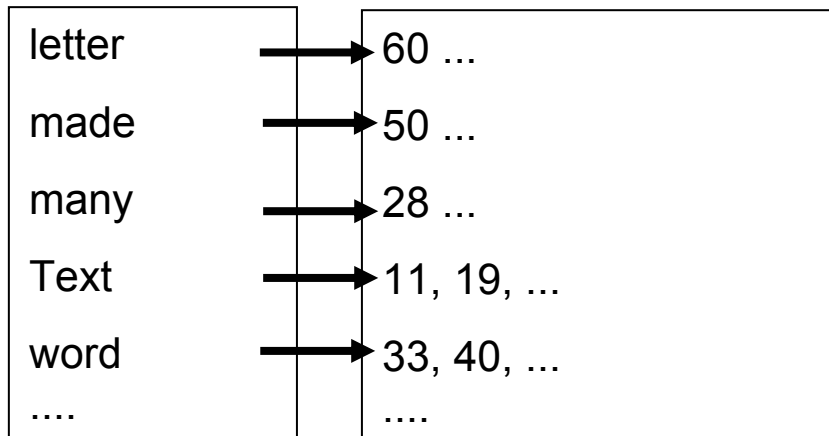
- **Example**

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.

**Text**

**Vocabulary**

**Occurrences**



An inverted list

Each element in a list points to a text position

An inverted file

Each element in a list points to a doc number

} difference:  
indexing granularity

# Inverted Files

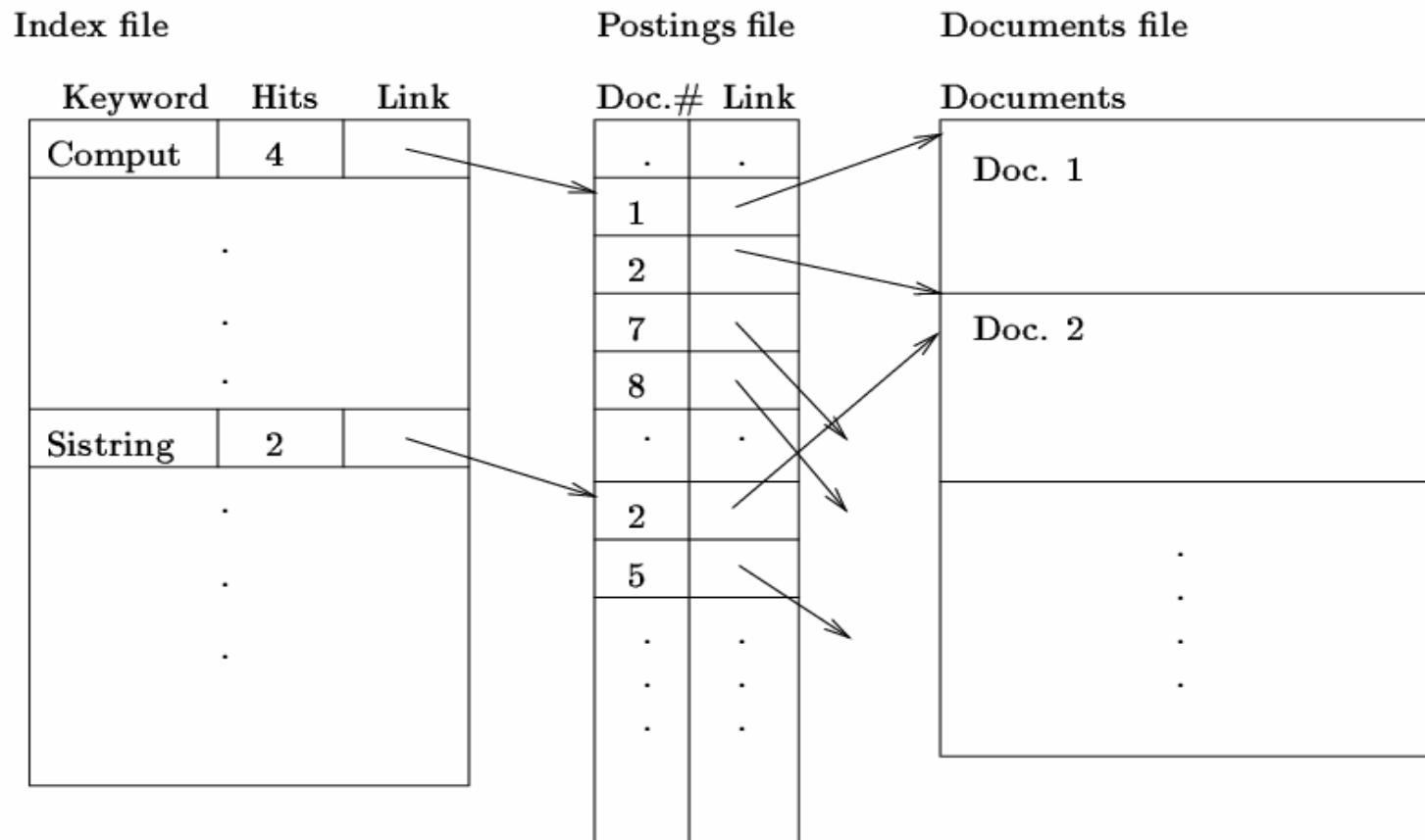
- **Implementation**

- Assume that the vocabulary (control dictionary) can be kept in main memory. Assign a sequential word number to each word
- Scan the text database and output to a temporary file containing the record number and its word number
- Sort the temporary file by word number and use record number as a minor sorting field
- Compact the sorted file by removing the word number. During this compaction, build the inverted list from the end points of each word. This compacted file (**postings file**) becomes the main index

.....  
 $d_5 w_3$   
 $d_5 w_{100}$   
 $d_5 w_{1050}$   
.....  
 $d_9 w_{12}$   
.....

# Inverted Files

- **Implementation (count.)**

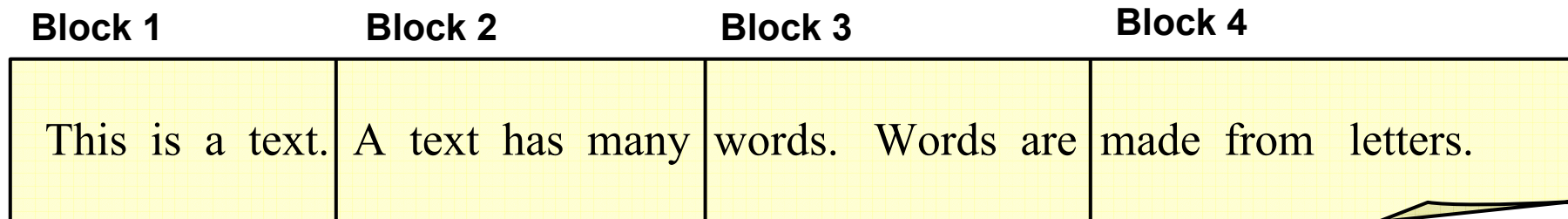


# Inverted Files: Block Addressing

- Features
  - Text is divided into blocks
  - The occurrences in the invert file point to blocks where the words appear
  - Reduce the space requirements for recording occurrences
- Disadvantages
  - The occurrences of a word inside a single block are collapsed to one reference
  - Online search over qualifying blocks is needed if we want to know the exact occurrence positions
    - Because many **retrieval units** are packed into a single block



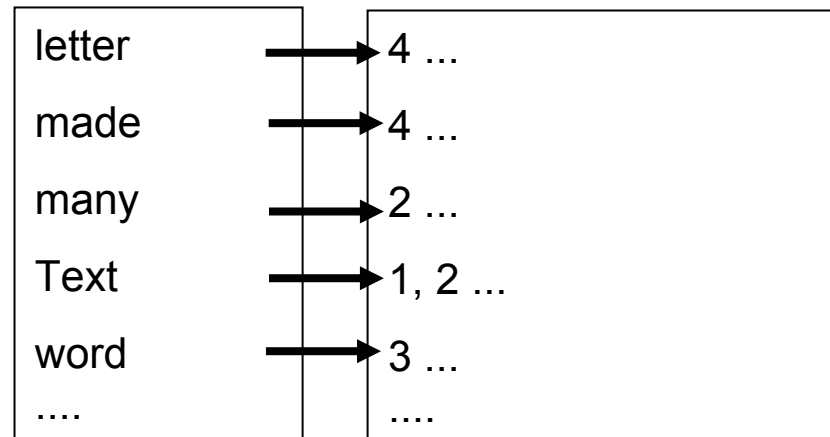
# Inverted Files: Block Addressing



**Text**

**Vocabulary**

**Occurrences**



**Inverted Index**

# Inverted Files: Some Statistics

- Size of an inverted file as approximate percentages of the size of the text collection

Index	Small Collection (1 Mb)		Medium Collection (200 Mb)		Large Collection (2 Gb)	
4 bytes/pointer Addressing Words	45%	73%	36%	64%	35%	63%
1,2,3 bytes/pointer Addressing Documents	19%	26%	18%	32%	26%	47%
2 bytes/pointer Addressing 64K blocks	27%	41%	18%	32%	5%	9%
1 byte/pointer Addressing 256 blocks	18%	25%	1.7%	2.4%	0.5%	0.7%

Stopwords are removed

Stopwords are indexed

# Inverted Files: Searching

- Three general steps
  - **Vocabulary search**
    - Words and patterns in the query are isolated and searched in the vocabulary
    - Phrase and proximity queries are split into single words
  - **Retrieval of occurrences**
    - The lists of the occurrences of all words found are retrieved
  - **Manipulation of occurrences** *intersection, distance, etc.*
    - For phrase, proximity or Boolean operations
    - Directly search the text if block addressing is adopted

# Inverted Files: Searching

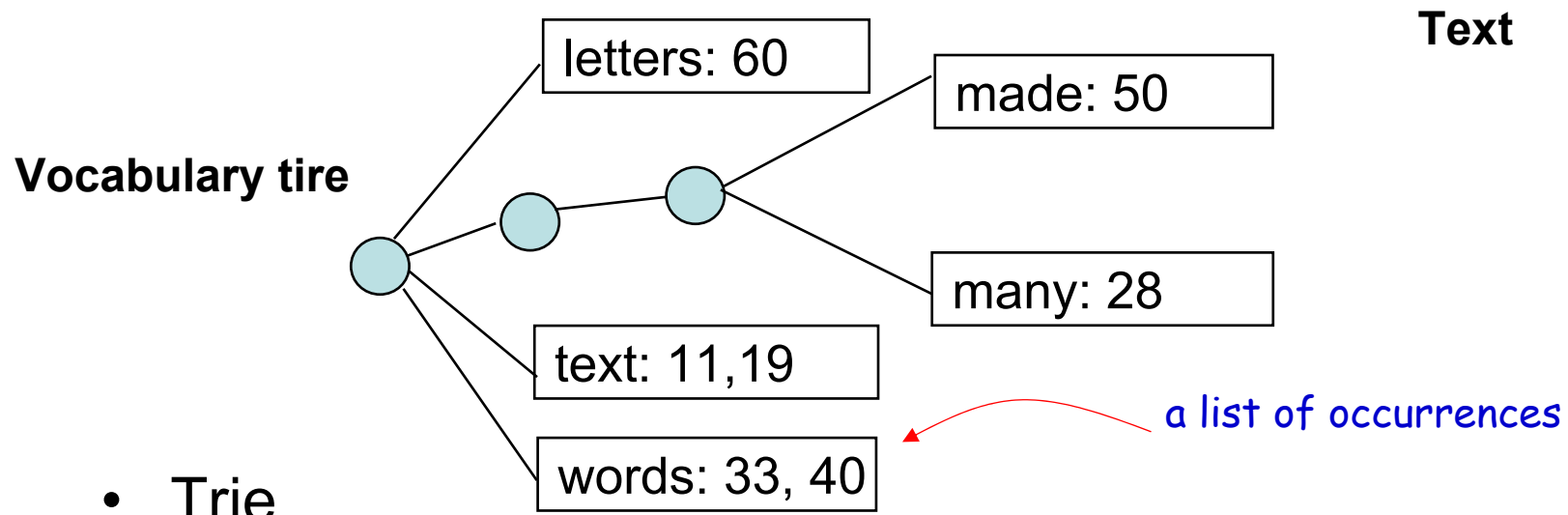
- Most time-demanding operation on inverted files is the merging or intersection of the lists of occurrences
  - E.g., **for the context queries**
    - Each element (word) searched separately and a list (occurrences for word positions, doc IDs, ..) generated for each
- The lists of all elements traversed in synchronization to find places where all elements appear in sequence (for a phrase) or appear close enough (for proximity)

*An expansive solution*

# Inverted Files: Construction

- The **trie** data structure to store the vocabulary

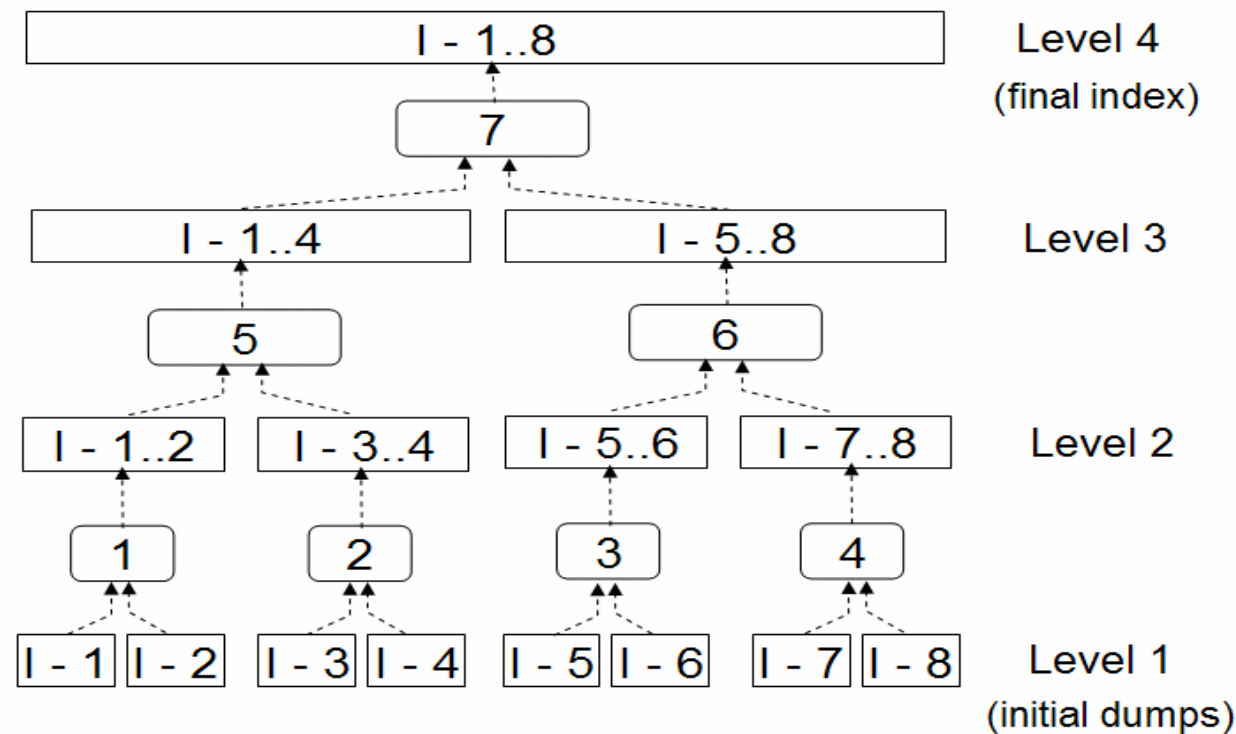
1 6 9 11 17 19 24 28 33 40 46 50 55 60  
This is a text. A text has many words. Words are made from letters.



- Trie
  - A digital search tree
  - A multiway tree that stores set of strings and able to retrieve any string in time proportional to its length
  - A special character is added to the end of string to ensure that no string is a prefix of another (words appear only at leaf nodes)

# Inverted Files: Construction

- Merging of the partial indices
  - Merge the sorted vocabularies
  - Merge both lists of occurrences if a word appears in both indices



# Signature Files

- **Basic Ideas**

- **Word-oriented index structures based on hashing**

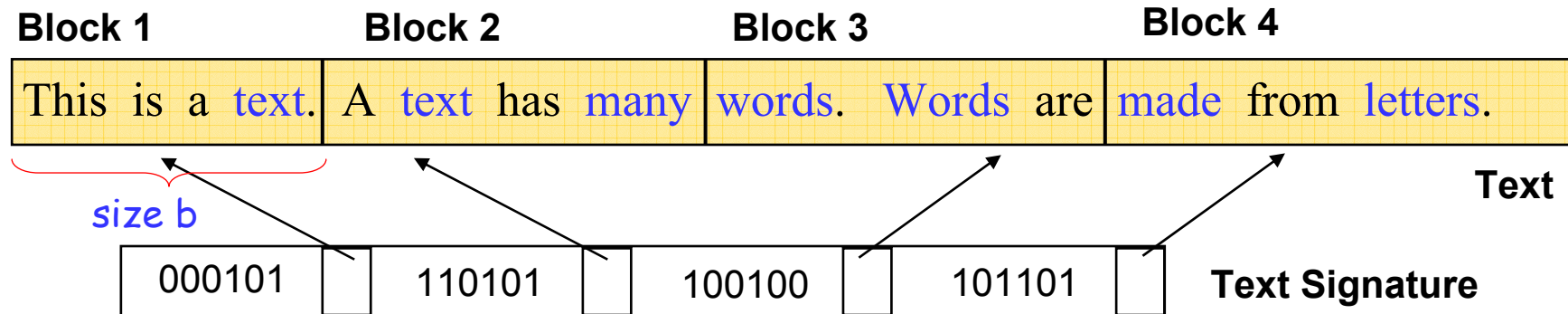
- A hash function (signature) maps words to bit masks of  $B$  bits

- Divide the text into **blocks of  $b$  words** each

- **A bit mask of  $B$  bits** is assigned to each block by **bitwise ORing the signatures of all the words in the text block**

- A word is presented in a text block if all bits set in its signature are also set in the bit mask of the text block

# Signature Files



## Signature functions

$h(\text{text})$	=	000101
$h(\text{many})$	=	110000
$h(\text{words})$	=	100100
$h(\text{made})$	=	001100
$h(\text{letters})$	=	100001

size B

## Stop word list

this
is
a
has
are
from
.....

- The text signature contains
  - Sequences of bit masks
  - Pointers to blocks



# Signature Files

- **False Drops** or False Alarms
  - All the corresponding bits are set in the bit mask of a text block, but the query word is not there
  - E.g., a false drop for the index “letters” in block 2
- **Goals** of the design of signature files
  - Ensure the probability of a false drop is low enough
  - Keep the signature file as short as possible

tradeoff

# Signature Files: Searching

- **Single word queries**

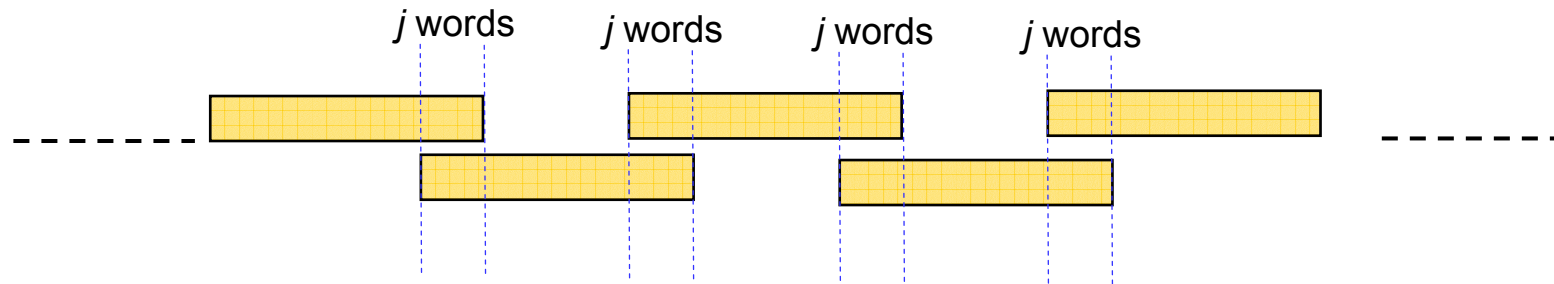
- Hash each word to a bit mask  $W$
- Compare the bit mask  $B_i$  of all text block (linear search) if they contain the word ( $W \& B_i == W$  ?)
  - **Overhead:** online traverse candidate blocks to verify if the word is actually there

- **Phrase or Proximity queries**

- The bitwise OR of all the query (word) masks is searched
- The candidate blocks should have the same bits presented “1” as that in the composite query mask
- **Block boundaries should be taken care of**
  - For phrases/proximities across two blocks

# Signature Files: Searching

- **Overlapping blocks**



- **Other types of patterns** (e.g., prefix/suffix strings,...) are not supported for searching in this scheme

- **Construction**

- Text is cut in blocks, and for each block an entry of the signature file is generated
  - Bitwise OR of the signatures of all the words in it
- Adding text and deleting text are easy

# Signature Files: Searching

- **Pros**

- Pose a low overhead (10-20% text size) for the construction of text signature
- Efficient to search phrases and reasonable proximity queries (the only scheme improving the phrase search)

- **Cons**

- Only applicable to index words
- Only suitable for not very large texts
  - Sequential search
  - Inverted files outperform signature files for most applications

# Suffix Trees

- **Premise**

- Inverted files or signature files treat the text as a sequence of words
  - For collections that the concept of word does not exist, they would be not feasible (like genetic databases)

- **Basic Ideas**

- Each position in the text considered as a text suffix
  - A string going from that text position to the end of the text (arbitrarily far to the right)
- Each suffix (or called **semi-infinite string**, **sistring**) uniquely identified by its position
  - Two suffixes at different position are lexicographical different

A special null character  
is added to the strings' ends

# Suffix Trees

- **Basic Ideas (cont.)**

- **Index points:** not all text positions indexed
  - Word beginnings
  - Or, beginnings of retrievable text positions
- Queries are based on prefixes of sistrings, i.e., on any substring of the text

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

sistring 11: text. A text has many words. Words are made from letters.

sistring 19: text has many words. Words are made from letters.

sistring 28: many words. Words are made from letters.

sistring 33: words. Words are made from letters.

sistring 40: Words are made from letters.

sistring 50: made from letters.

sistring 60: letters.

# Suffix Trees

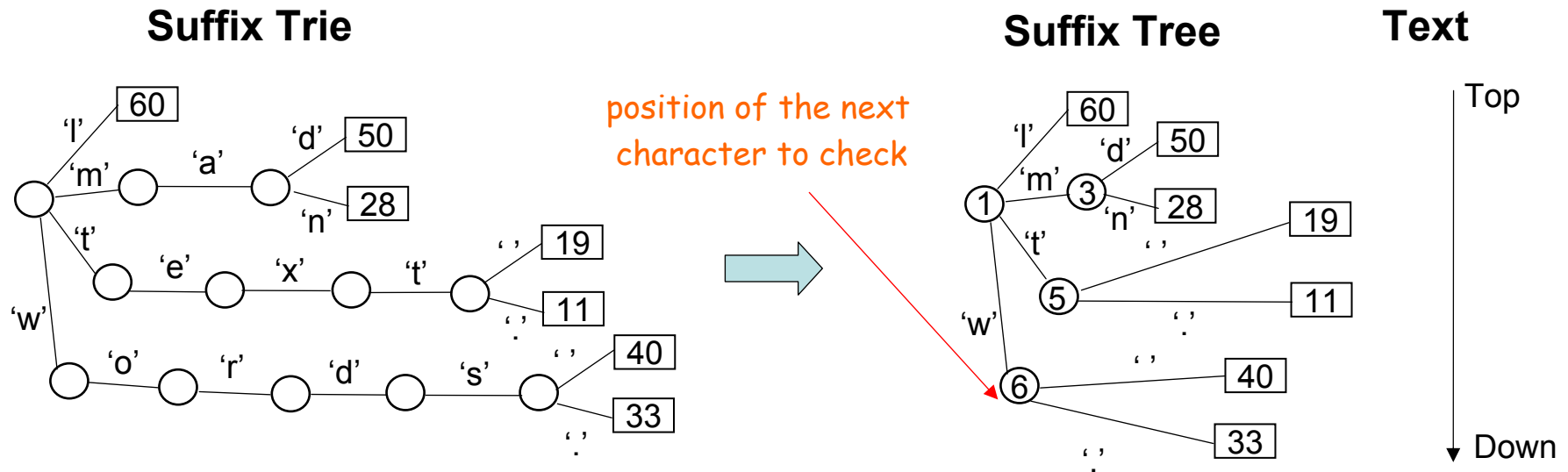
- **Structure**

- The suffix tree is a trie structure built over all the suffixes of the text
  - Points to text are stored at the leaf nodes
- The suffix tree is implemented as a **Patricia tree** (or **PAT tree**), i.e., **a compact suffix tree**
  - Unary paths (where each node has just one child) are compressed
  - An indication of next character (or bit) position to consider/check are stored at the internal nodes
    - Each node takes 12 to 24 bytes
    - A space overhead of 120%~240% over the text size

# Suffix Trees

- PAT tree over a sequence of characters

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.



If the query is "mo"

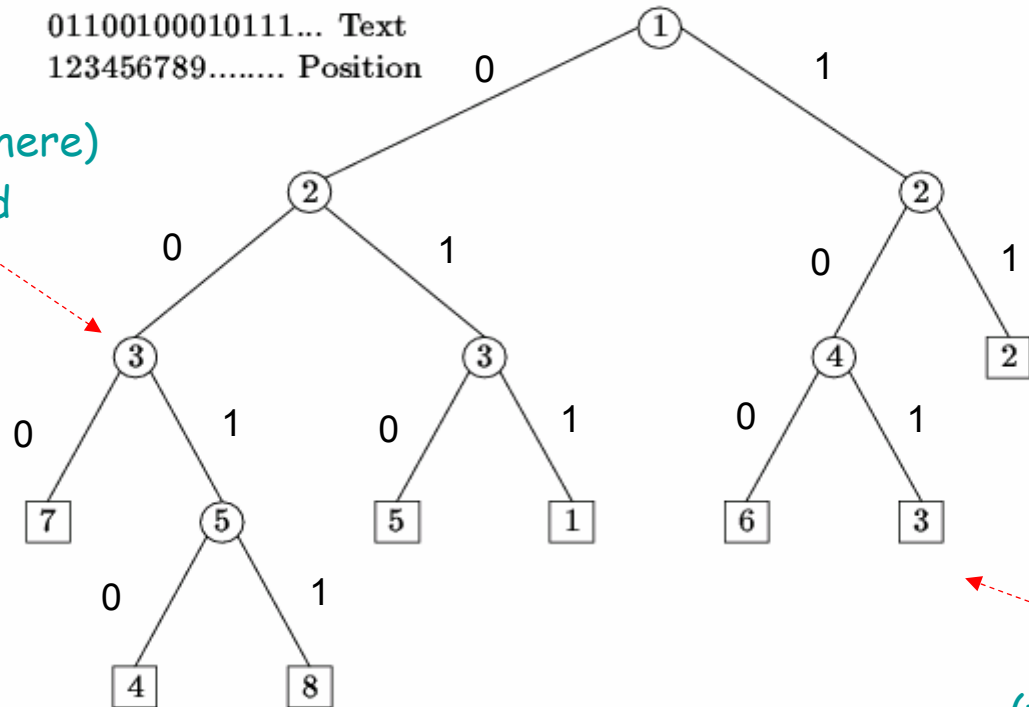


# Suffix Trees

- Another representation
  - PAT tree over a sequence of bits

The bit position of query used for comparison

- Absolute bit position (used here)
- Or the count of bits skipped (skip counter)



Internal nodes with single descendants are eliminated!

Example query: 00101

# Suffix Trees: Search

- Prefix searching

- Search the prefix in the tree up to the point where the prefix is exhausted or an external node reached

$O(m)$ ,  $m$  is the length in bits of the search pattern

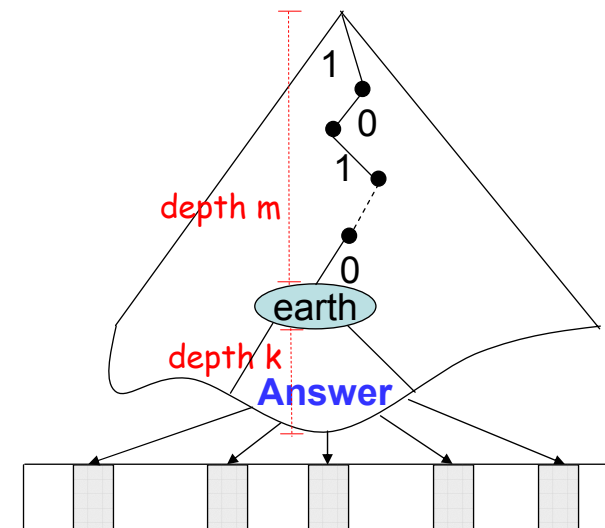
- Verification is needed

- A single comparison of any of the sistrings in the subtree

- If the comparison is successful, then all the sistrings in the subtree are the answer

$O(k \log k)$

- The results may be further sorted by text order



# Suffix Trees: Search

- Range searching
- Longest repetition searching
- Most significant or most frequent searching
  - Key-pattern/-word extraction

# Suffix Arrays

- **Basic Ideas**

- Provide the same functionality as suffix trees with much less space requirements
- The leaves of the suffix tree are traversed in left-to-right (or top-to-down here) order, i.e. lexicographical order, to put the points to the suffixes in the array
  - The space requirements the same as inverted files
- Binary search performed on the array
  - Slow when array is large

$O(n)$ ,  $n$  is the size of indices

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

60	50	28	19	11	40	33
----	----	----	----	----	----	----

one pointer stored for each indexed suffix

(~40% overhead over the text size)

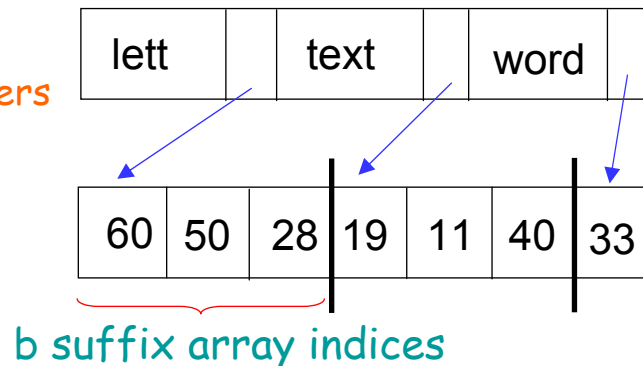
# Suffix Arrays: Supra indices

- Divide the array into blocks (may with variable length) and make a sampling of each block
  - Use the **first  $k$  suffix characters**
  - Use the **first word of suffix changes** (e.g., “text” (19) in the next example for nonuniformly sampling)
- Act as a first step of search to reduce external accesses (supra indices kept in memory!)

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.

The first 4 suffix characters are indexed

Suffixes sampled at fixed intervals



first  $k$  suffix characters

Supra-Index

Suffix Array

# Suffix Arrays: Supra indices

- Compare word (vocabulary) supra-index with inverted list

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.

letter	made	many	text	word
--------	------	------	------	------

Vocabulary  
Supra-Index

Suffixes sampled at  
fixed intervals

60	50	28	19	11	40	33
----	----	----	----	----	----	----

Suffix Array

60	50	28	11	19	33	40
----	----	----	----	----	----	----

Inverted List

major  
difference {

- Word occurrences in suffix array are sorted lexicographically
- Word occurrences in inverted list are sorted by text positions

# Suffix Trees and Suffix Arrays

- **Pros**

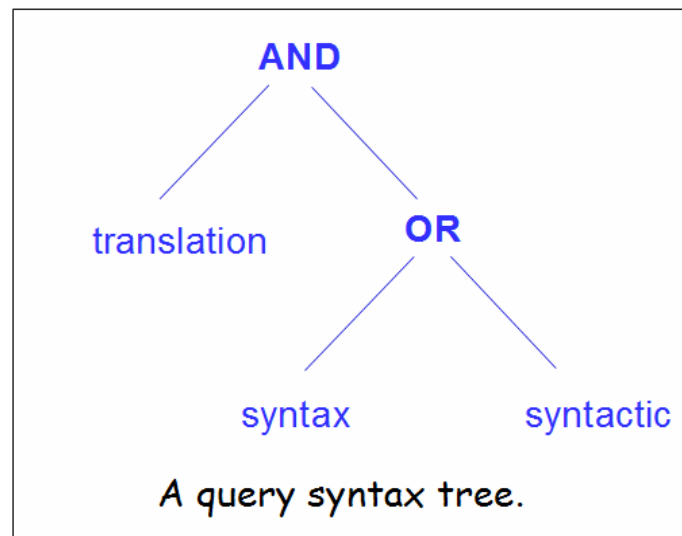
- Efficient to search more complex queries (phrases)
  - The query can be any substring of the text

- **Cons**

- Costly construction process
- Not suitable for approximate text search
- Results are not delivered in text position order, but in a lexicographical order

# Boolean Queries

- Set manipulation algorithms
  - Find the docs containing the basic queries given
  - The relevant docs are worked on by composition operators
  - Retrieve the exact positions of the matches and highlight them in the docs







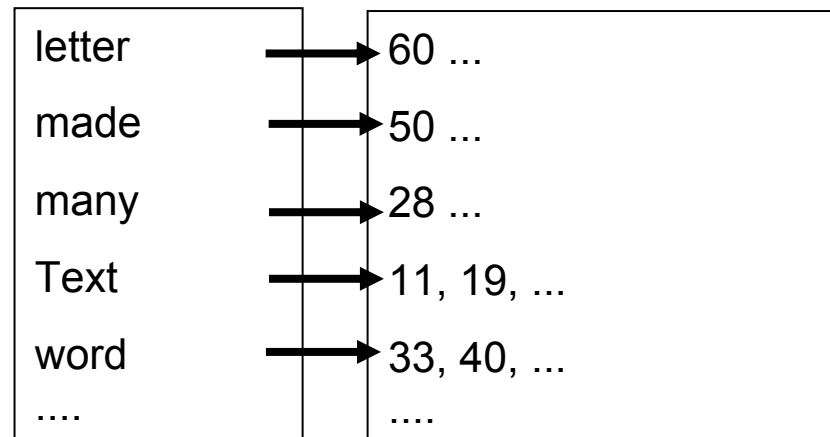


# Inverted Files

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

## Occurrences



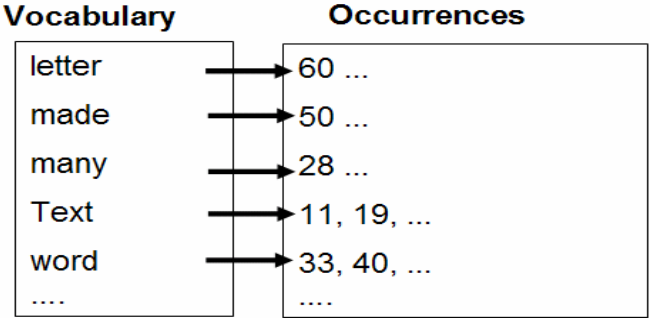
## An inverted list

Each element in a list points to a text position

## An inverted file

Each element in a list points to a doc number

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



An inverted list

Each element in a list points to a text position

An inverted file

Each element in a list points to a doc number

- Merging the partial indices

