Module V

CHAPTER 5

Indexing and Searching

Syllabus

Inverted files, Other indices for text, Boolean Queries, Sequential Searching, Pattern Matching, Structural

Queries, Compression; Multimedia IR: Indexing and Searching:- A Generic Multimedia indexing approach, , Automatic Feature extraction; Searching Web: Challenges, Characterizing the web, Search Engines. Browsing, Meta searches, Searching using Hyperlinks.

Self-learning Topics : Koha

This module will cover the main techniques required to implement the query operations.

- Initial concentration is on searching queries composed of words and on reporting the documents where they are found. It may also be necessary to know how many times a certain query appears in each document, as well as its precise locations within the text.
- Then, we focus on algorithms that deal with Boolean operations. Then, pattern matching and sequential search techniques are discussed.
- We consider compression methods and structured text last.

Searching the text sequentially is a simple way to find a basic query. When a text is not preprocessed, sequential or online text searching looks for instances of a pattern in the text. When the text is little (a few megabytes), online searching is appropriate.

Building data structures over the text (known as indices) is a second approach for accelerating the search. If the text collection is substantial and semi-static, it is worthwhile to create and maintain an index. Semi-static

collections can be updated on a regular basis (like daily), but they aren't thought to be able to handle, say, thousands of single-word insertions per second. This holds true not just for dictionaries or other slowly expanding literary works, but for the majority of real text databases. For example, it applies to journal archives and web search engines.

The best methods for medium large datasets (up to 200Mb), on the other hand, combine online and indexed searches.

5.1 INVERTED FILES

GQ. Describe the process of creating inverted index with example?

- A word-oriented approach for indexing a text collection to speed up searches is known as an inverted file (or inverted index).
- The vocabulary and the occurrences make up the two components of the inverted file structure.
- The collection of all the various words used in the text is the vocabulary. A list of all the text places where each such word appears is kept in memory for each such word.
- The term "occurrences" refers to the collection of all those lists (Fig. 5.1.1 shows an example). These positions may be used to refer to characters or words.

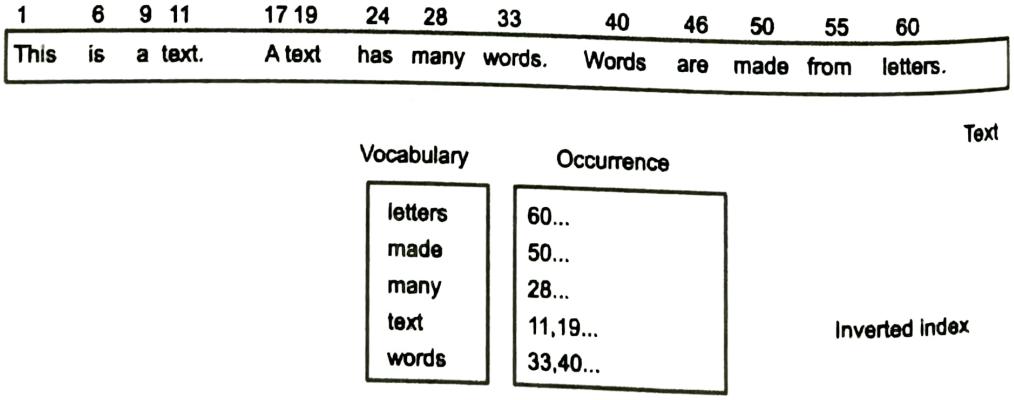
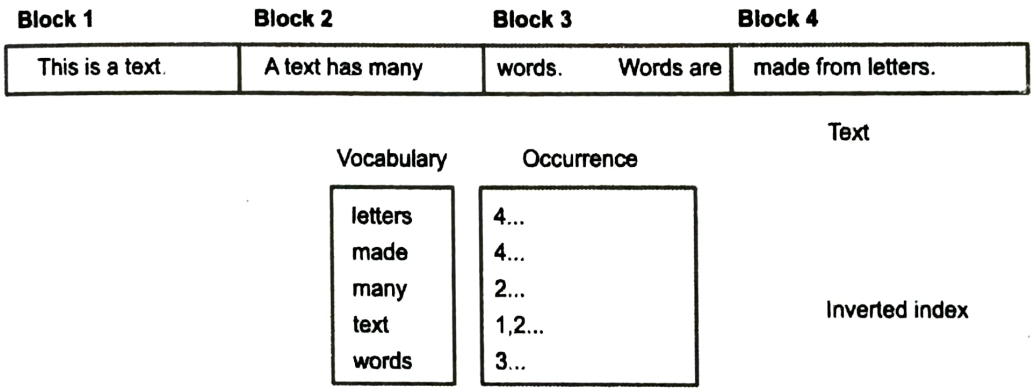


Fig. 5.1.1 : A sample text and an invested index built on it. The words are converted to lower-case and some are not indexed. The occurrences point to character positions in the text

- While character positions (i.e., the position I is the i-th character) make it easier to retrieve the matching text positions directly, word positions

(i.e., position I corresponds to the i-t, h word) simplifies phrase and proximity queries. The occurrences call for a lot more space.

- Each word used in the text is used only once in that structure, therefore the extra space is O(n).
- **Block addressing** is a technique used to decrease the amount of space needed. Blocks of text make up the text, and the occurrences indicate which blocks each word appears in (instead of the exact positions).
- The term "full inverted indices" refers to the traditional indices that pinpoint exact occurrences. By employing block addressing, all instances of a word within a single block are condensed into a single reference, allowing the pointers to be smaller because there are fewer blocks than places (see Fig. 5.1.2).
- This method yields indicators with a mere 5% overhead over the text size.



(1E2)**Fig. 5.1.2 : The sample text split into four blocks, and an inverted index using block addressing built on it. The occurrences denote block numbers. Notice that both occurrences of 'words' collapsed into one**

- The document addressing index assumes that we are pointing to 10 Kb-sized documents (and the necessary number of bytes per pointer, i.e. one, two, and three bytes, depending on text size).
- The block addressing index assumes that regardless of the text size, we employ 256 or 64 K blocks (one or two bytes per pointer).
- x Compression can be used to greatly minimise the amount of space that pointers occupy. We estimate that there are 11.5 characters between each non-stopword and that 45 % of all words are stop-words.

Table 5.1.1 : Sizes of an inverted file as approximate percentages of the size the whole text collectioin. Four granularities and three collections are considered. For each collection, the right column considers that stopwards are not indexed while the left column considers that all words are indexed.

Index	Small collection (1 Mb)		Medium collection (200 Mb)		Large collection (2 Gb)	
Addressing words	45 %	73 %	36 %	64 %	35 %	63 %
Addressing documents	19 %	26 %	18 %	32 %	26 %	47 %
Addressing 64 K blocks	27 %	41 %	18 %	32 %	5 %	9 %
Addressing 256 blocks	18 %	25 %	1.7 %	2.4 %	0.5 %	0.7 %

- The blocks can be defined using the text collection's natural division into files, documents, Web pages, or other categories, or they can be defined as fixed-size blocks that impose a logical block structure on the text database.

☞ Searching on an inverted index

There are three general steps in the search algorithm for an inverted index.

(1) **Vocabulary search :** The query's words and pattern are picked out and searched in the vocabulary. Take note that phrases and proximity queries are split into single words.

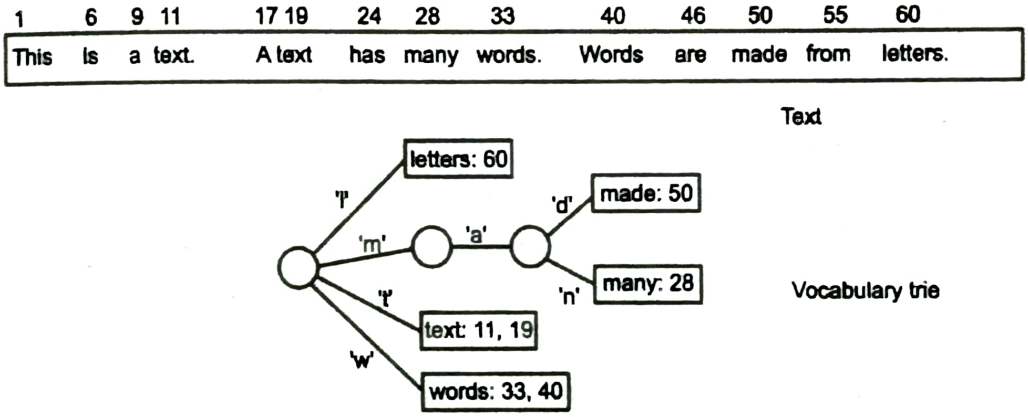
(2) **Retrieval of occurrences :** The lists of the occurrences of all the words that were discovered are retrieved.

(3) **Manipulation of occurrences :** To resolve phrases, proximity, or Boolean operations, the occurrences are analysed. If block addressing is employed, it might be necessary to perform a direct text search to locate the information that the occurrences are missing (e.g., exact word positions to form phrases).

- Practical data demonstrates, for instance, that the amount of text traversed and the space requirement can both be close to O (n0 85). We can therefore have sublinear search times with sublinear space needs because to inverted indices. The other indices do not allow for this.
- The following search times on our reference machine for a full inverted index constructed on 250 Mb of text are obtained: Simple word searches took 0.08 seconds, while phrase searches took 0.25 to 0.35 seconds (from two to five words).

☞ Construction of an Inverted Index

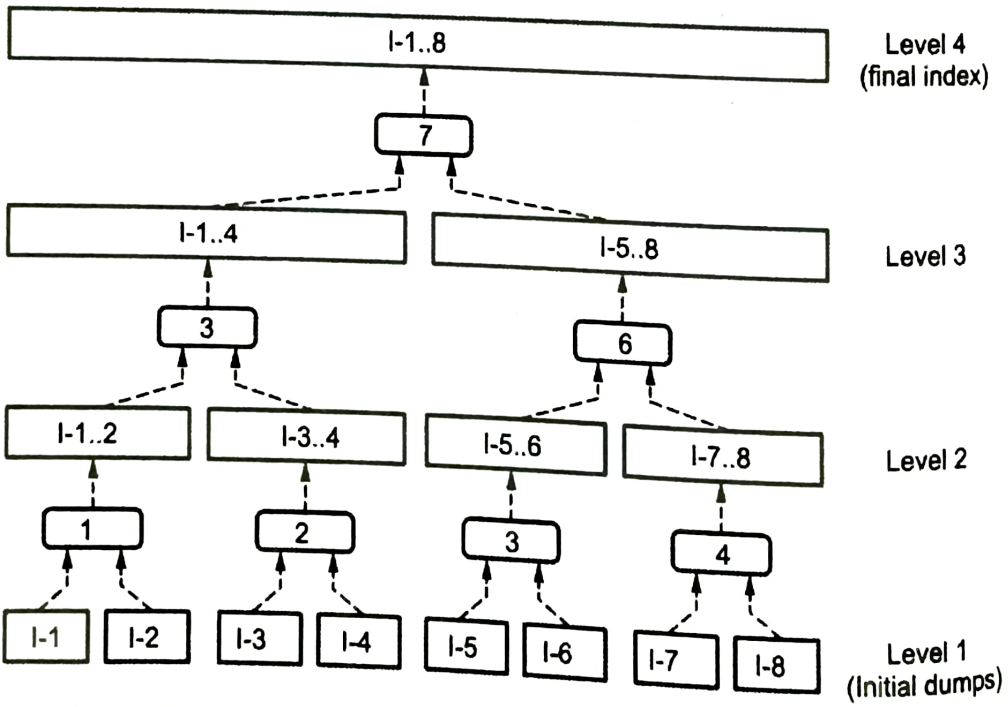
- An inverted index is reasonably easy to create and maintain. An inverted index on a text with n characters can theoretically be created in O(n) time.
- A trie data structure is used to store all of the vocabulary that is currently known, listing each word's instances for each word (text positions). Every word in the text is read and searched up in the dictionary.
- It is added to the trie with an empty list of occurrences if it is not discovered. The new place is added to the end of its list of occurrences once it is in the trie. This method is demonstrated in Fig. 5.1.3.



(1E3)**Fig. 5.1.3 : Building an inverted index for the sample text**

- The aforementioned approach is impractical for lengthy texts since it requires more memory than the index allows.
- The performance of the algorithm will be significantly hampered by a paging mechanism. We provide a substitute that is quicker in actual use.
- The previously mentioned procedure is employed until the main memory is exhausted.

- The portinf index fi collected thus far is written to disk and wiped from main memory when no more memory is available, and then the remainder of the text is continued.
- And finally, there are a few partial indices I; on disk. Then a hierarchical merge of these indices is performed.
- The index Z is created by combining the indices Z and J2, and the indexes 23 and 24 result in J3..4 I and so on.
- The resulting partial indices have about doubled in size. The merging process continues at the following level, connecting the index JI.2 with the index 3..4 to generate Jr..4 once all the indices at this level have been combined in this manner.
- As seen in Fig. 5.1.4, this continues until there is just one index that contains the whole text.



(1E4)Fig. 5.1.4 : Merging the partial indices in a binary fashion. Rectangles represent partial indices, while rounded rectangles represent merging operations. The numbers inside the merging operations show a possible merging order

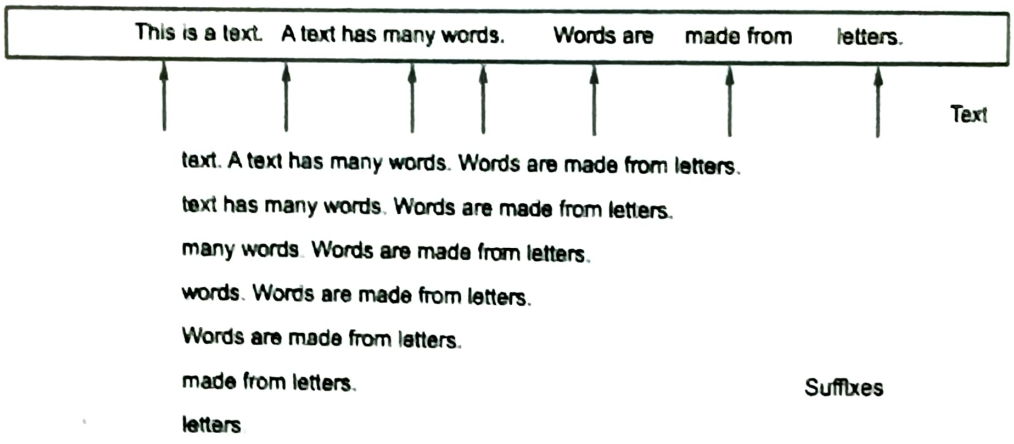
- It is possible to merge more than two indices at once. Even if the complexity is unaffected, efficiency is increased because there are fewer merging levels.

5.2 OTHER INDICES FOR TEXT

GQ. What are Suffix Trees. Discuss the searching in Suffix trees?
GQ. Describe the construction of Suffix Arrays for Large Texts?
GQ. Write in short Signature files.

Suffix Trees and Suffix Arrays

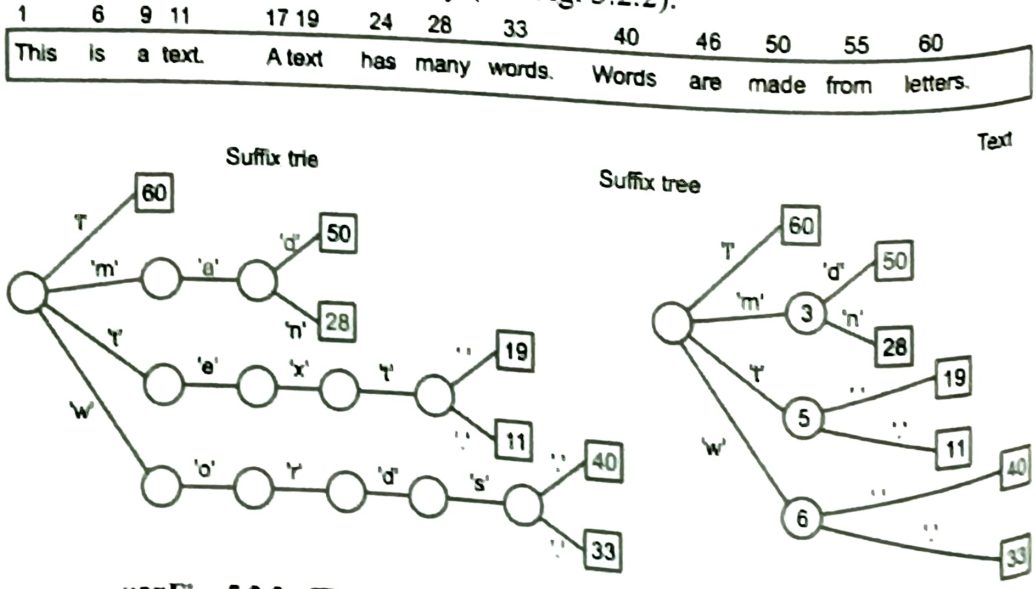
- A space-efficient way to implement suffix trees is via suffix arrays.
- With this kind of index, we can effectively respond to increasingly complicated requests.
- Its primary flaws are its expensive construction process, the requirement that the text be available at the time of the query, and the fact that the results are not delivered in text position order.
- This structure can be used to index every text character as well as only words (without stop words) as an inverted index.
- Because of this, it can be used for a larger range of applications, such genetic databases.
- The text is viewed by this index as one long string.
- Each text position is regarded as a text suns (i.e., a string that goes from that text position to the end of the text).
- It is obvious that two suffixes with distinct starting positions are different lexicographically (assume that a character smaller than all the rest is placed at the end of the text). Thus, the position of each suffix serves as a unique identification.
- Not every position of text must be indexed.
- From the text, index points are chosen that point to the beginning positions in the retrievable text.
- For instance, to have functionality comparable to inverted indices, it is feasible to index only word starts.
- It is impossible to retrieve elements that are not index points (as in an inverted index it is not possible to retrieve the middle of a word). This is seen in Fig. 5.2.1.



(1E5)**Fig. 5.2.1 : The sample text with the index points of interest marked. Below, the suffixes corresponding to those index points**

☞ Structure of Suffix Tree

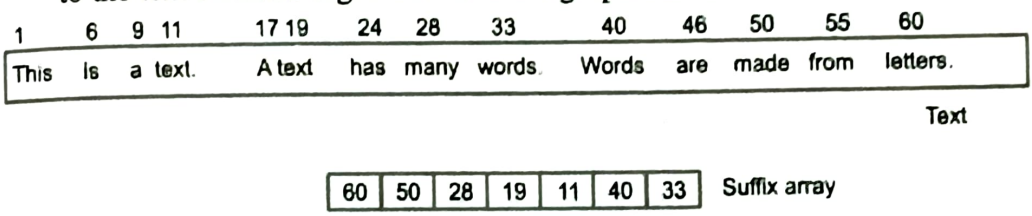
- A suffix tree is essentially a trie data structure constructed over all of the text's suffixes.
- The leaf nodes store the pointers to the suffixes.
- This trie is compressed into a Patricia tree to maximise space usage.
- Compressing unary paths—paths where each node has just one child—is required for this.
- The nodes that root a compressed path keep a record of the next character position to take into account.
- When unary paths are absent, the tree has 0(n) nodes rather than the trie's worst-case O(n 2) complexity (see Fig. 5.2.2).



(1E6)**Fig. 5.2.2 : The suffix trie and suffix tree for the sample text**

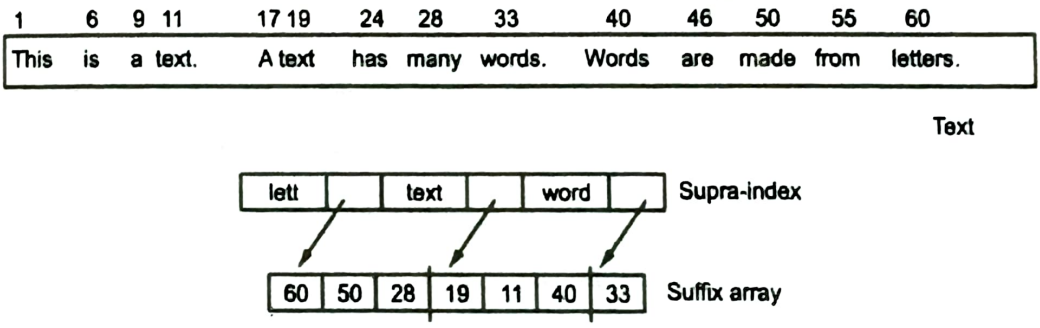
- The lack of space in this structure is a concern.

- Each node of the trie requires 12 to 24 bytes, depending on how it is implemented; as a result, even if only word beginnings are indexed, a space overhead of 120% to 240% over the text size is generated.
- With significantly less space needed, suffix arrays essentially offer the same functionality as suffix trees.
- All of the text's suffixes are recovered in lexical order if the leaves of the suffix tree are traversed from top to bottom (left to right in our figures).
- As seen in Fig. 5.2.3, a suffix array is just an array with all the pointers to the text suffixes organised in lexicographical order.



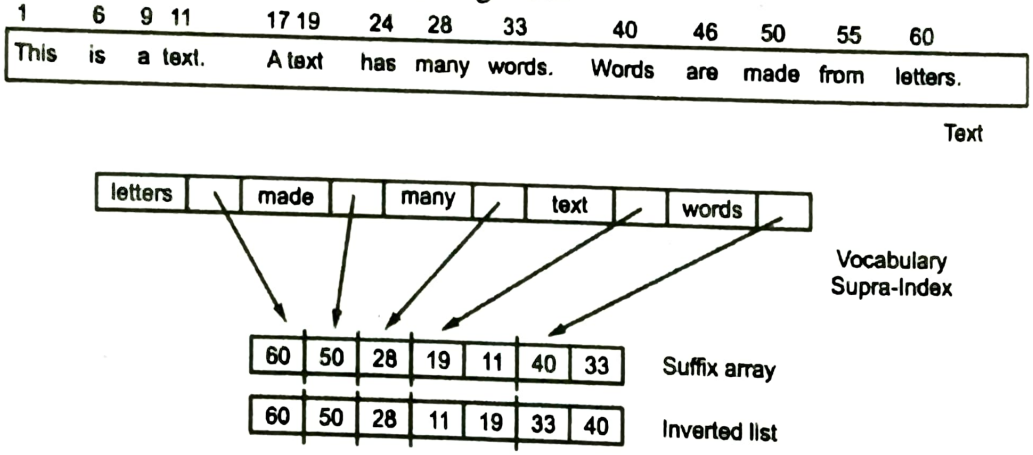
(1E7)**Fig. 5.2.3 : The suffix array for the sample text**

- Binary searches can be performed on suffix arrays by comparing the contents of each pointer.
- The number of random disk accesses can cause this binary search to perform poorly if the suffix array is huge. T
- he usage of supra-indices over the suffix array has been suggested as a solution to this problem.
- The simplest supra-index is nothing more than a sampling of one of b entries in a suffix array, with the first Z characters of each sample being kept in the supra-index.
- The initial step of the search is therefore to use this supra-index to minimise external accesses. A case is shown in Fig. 5.2.4.



(1E8)**Fig. 5.2.4 : A supra-index over our suffix array. One out of three entries are sampled, keeping their first four characters. The points (arrows) are in fact unnecessary**

- The only significant difference between this structure and an inverted index is that in a suffix array, the occurrences of each word are sorted lexicographically by the text that follows the word rather than by text position as they are in an inverted index.
- This connection is shown in Fig. 5.2.5.



(1E9)**Fig. 5.2.5 : Relationship between our inverted list and suffix array with vocabulary supra-index**

☞ Searching In a Suffix Tree

- Many fundamental patterns, including words, prefixes, and phrases, can be searched in O(m) time via a straightforward trie search if a suffix tree on the text can be afforded.
- Suffix trees, however, are not useful for lengthy texts, as stated. By using a binary search rather than a trie search, suffix arrays, on the other hand, can carry out the identical search operations in O(log n) time.
- Unfortunately, the binary search for suffix arrays is done on disk, where accesses to (random) text places necessitate a seek operation that spans the text's disk tracks.
- The size of a random seek is 0(n), hence the search takes 0(n log n) time. Any binary search procedure begins with the usage of supra-indices to address this issue.
- The search begins in the supra-index, which often fits in main memory, to avoid making O(log n) random visits to the text (plus the suffix array) on disk.
- After this search is finished, the binary search is completed, and the suffix array block that is situated between the two chosen samples is brought into memory (performing random accesses to the text on disk).

- As a result, disk searches now take just around 25% as long as before. A further reduction of between 40% and 60% is possible with modified binary search approaches that compromise the exact partition in the array's middle while taking the current disk head position into consideration.

☞ Construction in Main Memory

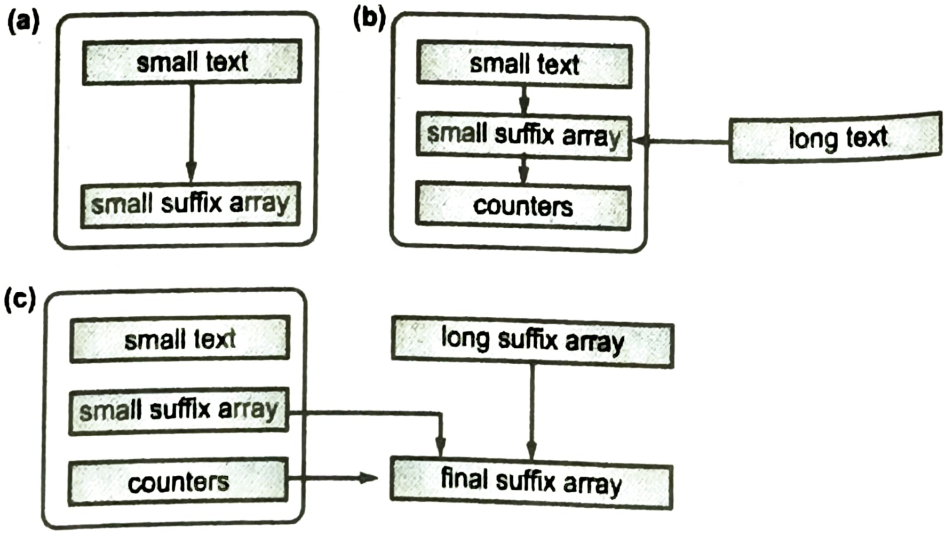
- In O(n) time, a suffix tree for a text with n characters can be constructed. But the algorithm struggles if the suffix tree does not fit in main memory, which is particularly demanding given how much space suffix trees require.
- The pointers are gathered in ascending text order and then simply sorted by the text they point to because the suffix array is nothing more than the set of pointers that have been lexicographically sorted.
- Be aware that the corresponding text places must be accessible in order to compare two elements in a suffix array.
- These accesses are largely arbitrary. Therefore, main Memory is required for both the suffix array and the text. String comparisons in this approach cost O(n log n) units.

☞ Construction of Suffix Arrays for Large Texts

- The issue of large text databases not fitting in main memory still exists.
- Applying an external memory sorting method might be possible.
- The text is accessed at random locations on the disk for each comparison, though.
- The sorting process' performance will be significantly reduced as a result.
- We describe an algorithm created specifically for large texts.
- Divide the text into manageable-for-main-memory blocks. Then, create the suffix array for each block in main memory and combine it with the remainder of the array that was already created for the previous paragraph.
- That is :
    - o build the suffix array for the first block,
    - o build the suffix array for the second block,

- merge both suffix arrays,
- build the suffix array for the third block,
- merge the new suffix array with the previous one,
- build the suffix array for the fourth block,
- merge the new suffix array with the previous one,
- ...and so on.

- How to combine a small suffix array with an already-built large suffix array is the challenging part (just built).
- The answer is to first calculate the distance between each pair of elements in the small array and the distance between each element in the large array, and then use that knowledge to merge the arrays without referencing the text.
- The quantity between each pair of spots in the small suffix array in the large text's suffixes is what we need to know.
- We create counters that keep track of this data. The large suffix array is not used to compute the counters. The huge array's associated text is sequentially read into the main memory.
- The small suffix array is searched for each suffix in that text (in main memory). We just increase the relevant counter once we have identified the inter-element position where the suffix is located. This procedure is demonstrated in Fig. 5.2.6.



(1E10)**Fig. 5.2.6 : A step of the suffix array construction for large texts : (a) the local suffix array is built, (b) the counters are computed, (c) the suffix arrays are merged**

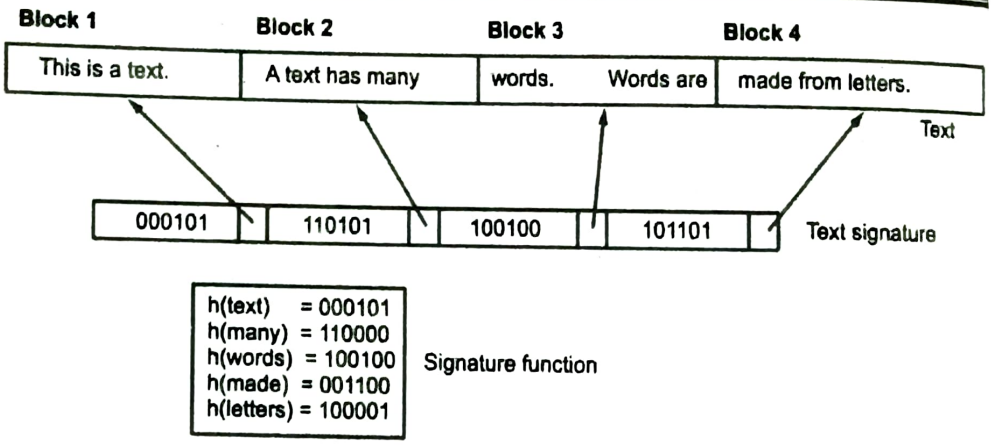
- Suffix array construction really costs more than inverted file construction. Through a fast final sequential pass over the suffix array, the supra-index is built.
- On the reference machine, indexing takes about 0.8 Mb/min for 250 Mb of text. The construction of inverted indices is five to ten times faster than this.

☞ Signature Files

- Hash-based word-oriented index structures are what make up signature files.
- At the expense of requiring a sequential search through the index, they have a low overhead (10 % to 20 % over the text size).
- Although their search complexity is linear (as opposed to sublinear as with the prior approaches), its constant is quite low, making the technique appropriate for texts that aren't particularly huge.
- However, for the majority of applications, inverted files function better than signature files.

☞ Structure of Signature Files

- The hash function (also known as "signature") in a signature file converts words to bit masks of B bits.
- The text is divided into blocks of b words each. A bit mask of size B will be assigned to each text block of size b.
- Bitwise ORing the signatures of each word in the text block yields this mask.
- As a result, the signature file is nothing more than a list of all blocks' bit masks (plus a pointer to each block).
- The key concept is that all of the bits set in a word's signature are set in the text block's bit mask if the word is present.



(1E11)**Fig. 5.2.7 : A signature file for our sample text cut into blocks**

- Bit masks with at least I set bits must be delivered by the hash function.
- A suitable model assumes that the mask's Z bits are set at random (with possible repetition).
- Let I/B be n. The chance that a certain bit of the mask is set in a word signature is

$$1 - (1 - 1/B)^{bl} \approx 1 - e^{-b\alpha}$$

because each of the b words sets I bits at random, the probability that the I random bits set in the query are likewise set in the mask of the text block is.

$$(1 - e^{-b\alpha})^{\alpha B}$$

which is minimized for $\alpha = ln\ (2) / b$.

- The false drop probability under the optimal selection

$$l = B\ ln\ (2) / b \text{ is } (1/2^{ln\ (2)})^{B/b} = 1/2^{l}$$

☞ Searching in Signature Files

- By hashing a word to a bit mask IV and comparing it to the bit masks B; of all the text blocks, a single word can be searched.
- An online traversal must be done for each and every potential text block to ensure the term is indeed present. The same as with inverted files, this traversal cannot be avoided.
- In this method, no other kinds of patterns may be searched. However, the method responds to realistic proximity queries and search keywords more efficiently.

- A small 2.8 Mb database's queries took 0.42 seconds to complete. Using extrapolation to the state of technology now, we discover that the performance should be close to 20 Mb/sec , making the example of 250 Mb of text take 12 seconds, which is a long time.

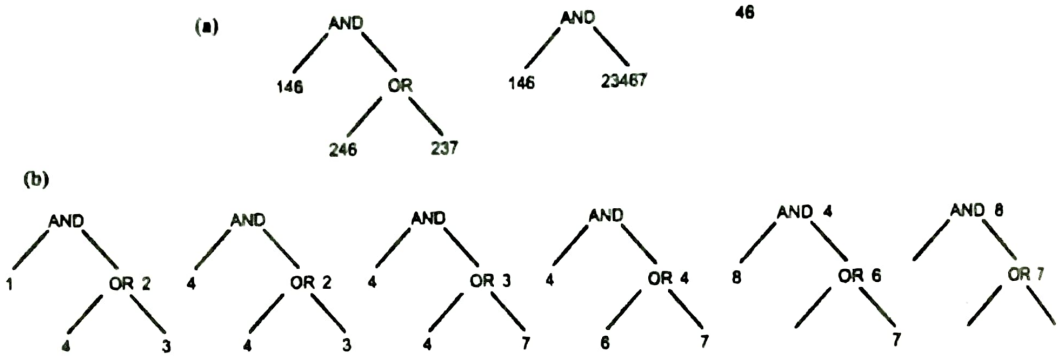
☞ Construction of Signature Files

- Building a signature file is rather simple. The text is simply divided into blocks, and an entry for each block is created in the signature file. This item represents the bitwise OR of all of the block's word signatures.
- It is also simple to add text because all that is required is to keep adding records to the signature file. By erasing the relevant bit, text can be deleted.
- For Instance, A separate file might be created for each bit of the mask, with all the first bits in one file, all the second bits in another, etc.
- As a result, searching the disk for a query takes less time since only the files corresponding to the I bits set in the query need to be visited.

⏭ 5.3 BOOLEAN QUERIES

GQ. Compare Boolean queries & Natural Language queries?

- Boolean queries operate on sets of results, which calls for the application of the manipulation algorithms.
- Composition operators must work on the relevant documentation after the leaves of the query syntax tree are solved.
- Typically, a search is conducted in three stages :
  (1) The first stage decides which documents to categorize;
  (2) The second stage decides the relevance of the clarifying documents so that they are presented to the user in an appropriate way; and
  (3) The third stage retrieves the exact positions of the matches so that they are highlighted in the documents that the user actually wants to see.
- This plan prevents wasting time working on documents that won't be read or classified until the very end (first phase) (second phase).



(1E12)Fig. 5.3.1 : Processing the internal nodes of the query syntax tree. In (a) full evaluation is used. In (b) we show lazy evaluation in more detail.

- The internal nodes of the query syntax tree further manage these sets of documents after the leaves have located the classifying sets of documents.
- Using identities like a OR (a AND b) = a, the tree can be algebraically optimized.
- The syntax tree can be evaluated in Full or Lazy form. In the full evaluation form, the entire result is generated after both operands have been fully received.
- Results from lazy evaluation are only offered when requested, and both operands must be provided with some data iteratively in order to get that result.
- Due to the knowledge of the sizes of the findings, full evaluation enables some optimizations.
- On the other hand, lazy evaluation enables the application to decide when to perform the task of acquiring new results, as opposed to stopping it for an extended period of time.
- Hybrid schemes are also possible. Fig. 5.3.1 demonstrates this

5.4 SEQUENTIAL SEARCHING

GQ. Define text search algorithm?

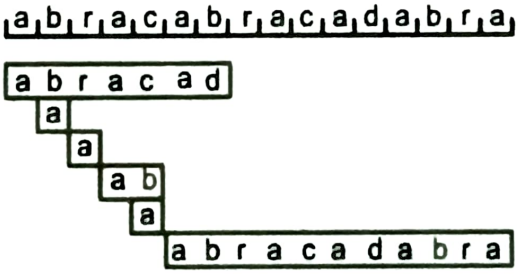
GQ. Describe about the techniques involved in software text search algorithm?

GQ. List out the differences between Boyer –Moore text search algorithm and Knuth-Pratt Morris algorithm?

- Now that the text has not been transformed into a data structure, let's discuss text search techniques.
- It is a fundamental component of several indexing strategies and, in some situations, the only choice.
- This section deals with exact string matching.
- Finding all the text locations where a pattern appears given a short pattern P of length m and a lengthy text T of length n is the exact string matching problem.
- Many common queries, including word, prefix, suffix, and substring searches, can be incorporated into this problem with few alterations.
- The first character from alphabet of size , with position 1, is taken as the basis for both the text and the pattern.
- Most case analyses presume that the text and patterns are random.

☞ Brute Force

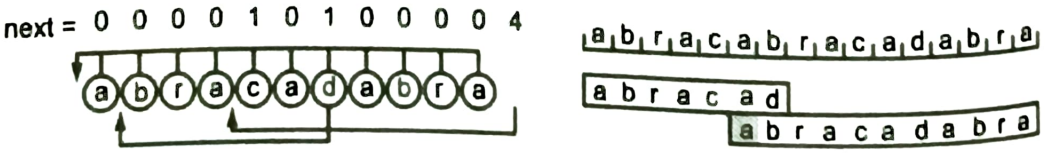
- The simplest algorithm is the brute-force (BF) algorithm.
- Simply attempting every pattern place in the text is all that is required.
- It checks to see if the pattern matches at each of these positions. Look at Fig. 5.4.1.
- The worst-case scenario for brute-force searching is O (mn) since there are 0 (n) text positions and each one is inspected at 0 (m) worst-case cost O (mn). However, it has an O (n) average case.
- This algorithm does not need any pattern preprocessing.
- The text is covered by a window that is m lengths long. The text in the window is compared against the pattern to see if they match.
- The window is then moved ahead. The primary way the algorithms vary is in how they check and move the window.



(1E13)**Fig. 5.4.1 : Brute-force search algorithm for the pattern 'abracadabra.' Squared areas show the comparisons performed**

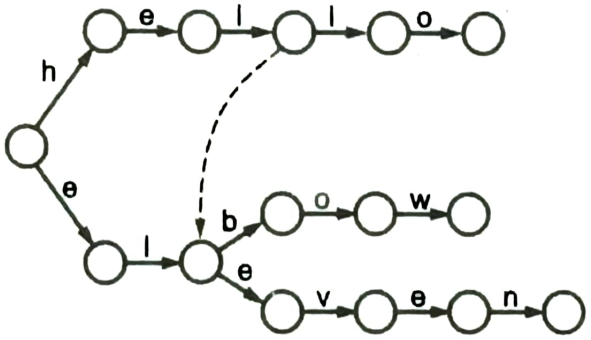
☞ Knuth-Morris-Pratt

- Although it is not significantly quicker than BF on average, the KMP method was the first to have linear worst-case behavior.
- A window is also slid over the text by this method. It does not, however, test every window position like BF does.
- Instead, it makes use of data from earlier checks. A number of pattern letters were compared to the text window after the window was examined to see if it fit the pattern or not, and all of them matched possibly with the exception of the last one compared.
- As a result, the pattern that matched the text becomes prey when the window needs to be adjusted.
- The method makes use of this knowledge to prevent trying window placements that are obvious mismatches.
- To create a table called next, the pattern is preprocessed in O (m) time and space.
- The longest proper prefix of $P_{1..j-1}$, which is also a suffix, is listed in the following table at position j. The characters that follow the prefix and suffix are different.
- Therefore, if the characters up to character j-1 matched but the character j-th did not, the window positions j-next[J] + 1 can be safely skipped.
- For instance, if a text window matched the word "abracadabra" up to "abracab," five slots can be safely omitted because next[7J = 1.
- An instance is shown in Fig. 5.4.2.
- The key finding is that the pattern is the only thing on which this information depends; if the text in the window matched position j – 1, then the text matches the pattern.



(1E14)Fig. 5.4.2 : KMP algorithm searching 'abracadabra.' On the left, an illustration of the next function. Notice that after matching 'abracada' we do not try to match the lost 'a' with the first one since what follows cannot be a 'b'. On the right, a search example. Grayed areas show the prefix information reused.

- When it comes to matching a set of patterns, the Aho-Corasick algorithm can be seen as an extension of KMP.
- The patterns are set up in a data structure that resembles a trie.
- Each trie node indicates that it has matched a certain pattern's prefix (s). A broader collection of failure transitions takes the place of the following function.
- The trie's nodes are crossed by those transitions.
- Assuming that y is the longest prefix in the set of patterns and is also a proper suffix of x, a transition leaving from a node representing the prefix x leads to a node representing the prefix y.
- This is seen in Fig. 5.4.3.



(1E15)**Fig. 5.4.3 : Aho-Corasick trie example for the set 'hello,' 'elbow' and 'eleven' showing only one of all failure transitions.**

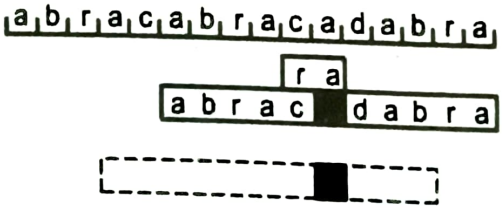
- This trie is constructed in O (m) time and space, along with its failure transitions (where m in the total length of all the patterns).
- No matter how many patterns are searched, the search time is O (n). It does at most 2n inspections, similar to KMP.

☞ Boyer-Moore Family

- The check inside the window can proceed backwards, is the foundation of BM algorithms.
- A suffix of the pattern has been compared and found to be equal to the text in the window when a match or mismatch is decided.
- This can be used in a manner that is quite similar to the next table of KMP, i.e., compute the next-to-last occurrence of $P_{j...m}$ inside P for each pattern position j. It is known as the "match heuristic."
- Together, this and something known as the "occurrence heuristic" are

used.

- The text character that caused the mismatch (if one occurred) must line up with the same character in the pattern after the shift, according to the rule.
- The heuristic that results in the largest shift is chosen.
- Consider searching for "abracadabra" in a text that begins with "abracababra," for example.
- After matching the suffix 'abra' the underlined text letter 'b' will create a mismatch.
- According to the match heuristic, a shift of 7 is safe because "abra" was matched.
- According to the occurrence heuristic, a shift of 5 is secure since the underlined "b" must match the pattern. Consequently, the pattern is 7 degrees off.
- Look at Fig. 5.4.4.
- This algorithm's preprocessing time and space are O (m + σ).
- Its average search time is O (n log(m)/m), which is 'sublinear' in that not every character is examined. However, the worst-case scenario is O (mn)



(1E16)**Fig. 5.4.4 : BM algorithm searching 'abracadabra.' Squared areas show the comparisons performed. Grayed areas have already been compared (but the algorithm compares them again). The dashed box shows the match heuristic, which was not chosen.**

5.5 PATTERN MATCHING

This section looks at complex patterns grouped as, searching allowing errors and searching for extended patterns.

String Matching Allowing Errors

- Given a short pattern P of length m, a lengthy text T of length n, and a maximum number of errors k, discover all the text spots where the pattern appears with a maximum of k errors.

- This issue is known as "approximate string matching."
- The Levenshtein distance is represented by this statement. It may be easily modified to search for full words that match the pattern with k errors.
- There are already a few solutions, however this issue is more recent than exact string matching. We sketch the primary approaches.

☞ Dynamic Programming

- Dynamic programming is the traditional approach for approximating string matching.
- Column by column, a matrix C [0..m, 0..n] is populated, where C [i, j] denotes the smallest number of errors required to match P1..i to a suffix of T1..j. The calculation is as follows:

$$C[0, j] = 0$$

$$C[i, 0] = i$$

$$C[i, j] = \text{if } (P_i = T_j) \text{ then } C[i-1, j-1]$$

$$\text{else } 1 + \min(C[i-1, j], C[i, j-1], C[i-1, j-1])$$

where a match is reported at text positions *j* such that $C[m, j] \leq k$

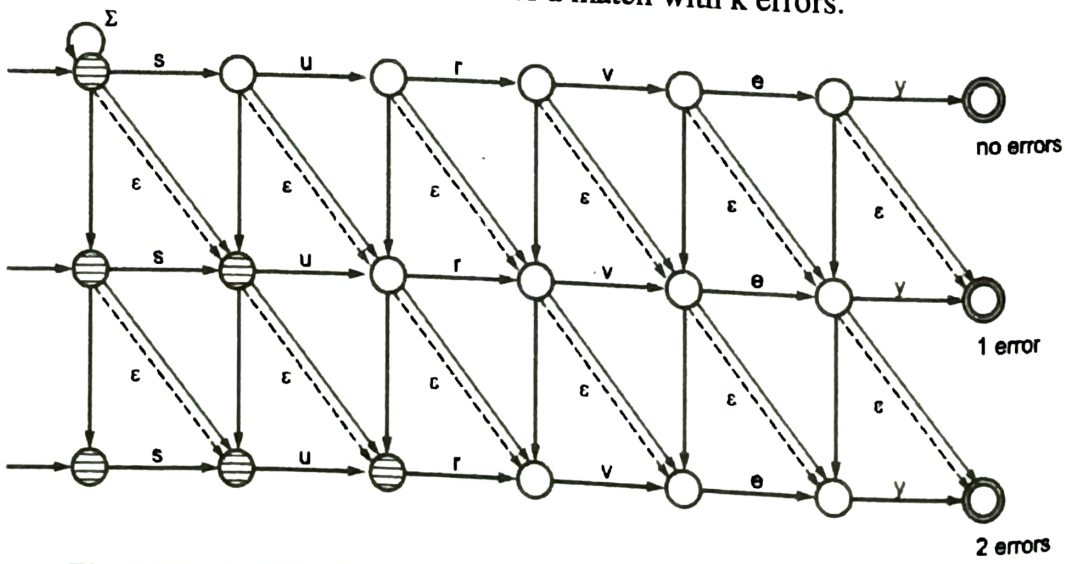
- The algorithm is therefore O (mn) time.
- It requires only the previous matrix column, hence it can be implemented in O (m) space. Its preprocessing took O (m) of time.
- This approach is demonstrated in Fig. 5.5.1.

		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

Fig. 5.5.1 : The dynamic programming algorithm search 'survey' in the text 'surgery' with two errors. Bold entries indicate matching positions.

☞ Automaton

- The fact that the issue can be reduced to a non-deterministic finite automaton is interesting (NFA).
- Take into account the NFA for k = 2 errors in Fig. 5.5.2.
- The number of errors shown in each row is indicated. 0 for the first, 1 for the second, and so on.
- Each column stands for completing the pattern up to a specific location.
- The automaton's states are changed as a new text character is read during each loop.
- Vertical arrows represent pattern insertions, solid diagonal arrows represent pattern re- placements, and dashed diagonal arrows represent pattern deletions.
- Horizontal arrows signify matching a character.
- Every time the (k + 1)th rightmost state is active, the automaton accepts a text position as the conclusion of a match with k errors.



(1E17)**Fig. 5.5.2 : An NFA for approximate string matching of the pattern 'survey' with two errors. The shaded states are those active after reading the text 'surgery'. Unlabelled transitions match any character.**

- It is simple to understand that when a state of the automaton becomes active, all the states of that column and higher rows also become active.
- Additionally, if we gather the smallest active rows in each column for a given text character, we may determine the current column of the dynamic programming process. This is seen in Fig. 5.5.2. (compare the

figure with Fig. 5.5.1).

- Making this automata deterministic is one option (DFA). The DFA can be very large even though the search phase is O (n).
- Bit-parallelism gives an alternate solution.

☞ Bit-Parallelism

- Bit-parallelism has been used to parallelize the NFA computation (without making it deterministic), achieving O (kmn/w) time in the worst case, and the dynamic programming matrix computation (achieving average complexity O (kn/w).
- These algorithms, which operate at 6 to 10 Mb per second on our reference computer, are currently the fastest ones and reach O (n) search times for short patterns.

☞ Filtering

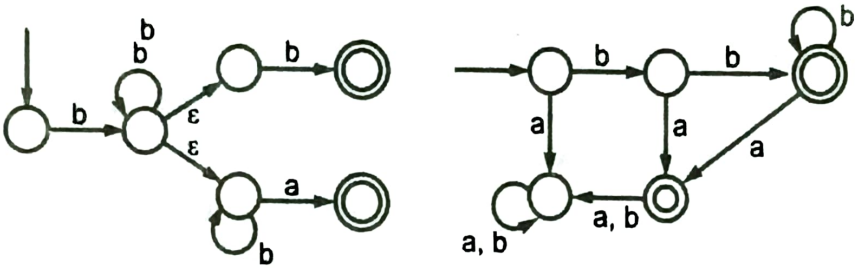
- Last but not least, various methods first filter the content to minimise the requirement for dynamic programming.
- Although the filtration is ineffective for higher error ratios, these algorithms frequently achieve 'sublinear' anticipated times.
- Filtration is predicated on the requirement that certain patterns, even in an approximate occurrence, must show with no errors.
- The quickest algorithm for low error levels is filtering-based: if the pattern is divided into k + 1 pieces, any approximate occurrence must have at least one of the pieces without errors as k errors cannot change all of the k + 1 pieces.
- Therefore, the search process starts with a multipattern exact search for the parts and then verifies any potential matching locations.

☞ Regular Expressions and Extended Patterns

- By creating an automaton that locates all instances of a general regular expression in a text, general regular expressions are searched.
- A non-deterministic finite automaton of size O (m), where m is the length of the regular expression, is initially constructed in this way.
- Making this automata deterministic is the traditional solution. Any regular expression can be searched in O (n) time by a deterministic

automaton.

- However, its size and building time can be exponential in m, i.e. O (m2m) (m2m).
- Look at Fig. 5.5.3.
- This algorithm uses the reference machine and operates at 6 Mb/sec, excluding preprocessing.



(a) and deterministic **(b) automata for the regular expression b b* (b | b* a)**

(1E18)**Fig. 5.5.3 : The non-deterministic**

- Regular expressions can be used to express extended patterns and be solved as before. However, as we saw with the expansions of accurate searching, it is frequently more effective to provide them with a particular answer (bit-parallel algorithms).
- Additionally, for optimum versatility, extended patterns can be used with approximation search.
- The bit-parallel technique is typically best suited to handle prolonged patterns.
- Real times for extended pattern searches and regular expressions using this method range from 2 to 8 Mb/sec.

5.6 STRUCTURAL QUERIES

- Each model has a significant impact on the search algorithms for structured text. In this part, we extract their common features.
- The first issue with this issue is how to store the structural data. In certain implementations, the structure is stored in an ad hoc index.
- This may be more effective and unrelated to the text's considerations. It does, however, necessitate more development and maintenance work.

- Other methods rely on the text being "tagged" to indicate the structure (i.e., strings that identify the structural elements). This is true for HTML text, but not for C code, since the marks are implicit and built-in to the language.
- By indexing and searching those tags as if they were words, the approach uses the same index to query content (like inverted files).
- This is frequently just as effective as an ad hoc index, and it integrates more easily into an existing text database.
- Furthermore, because the proper tags can be chosen at search time, the structure can be defined dynamically.
- Inverted files are more effective for achieving this purpose since they naturally offer the findings in text order, making it simpler to retrieve the structure information.
- On the other side, without an ad hoc index, it can be challenging to respond to some queries, including direct ancestry.
- A set of responses is generated once the content and structural components have been located using an index.
- Additional operations can be performed on those responses using the models, such as "select all areas in the left-hand argument which contain an area of the right-hand argument.
- "Generally speaking, this is resolved in a manner that is somewhat similar to the set manipulation approaches already described in section 5.4.
- It is not always possible to discover an evaluation method that has linear time with respect to the amount of the intermediate results, though, since the operations tend to be increasingly complex.

5.7 COMPRESSION

GQ. Explain the compression methods that enable text searching?

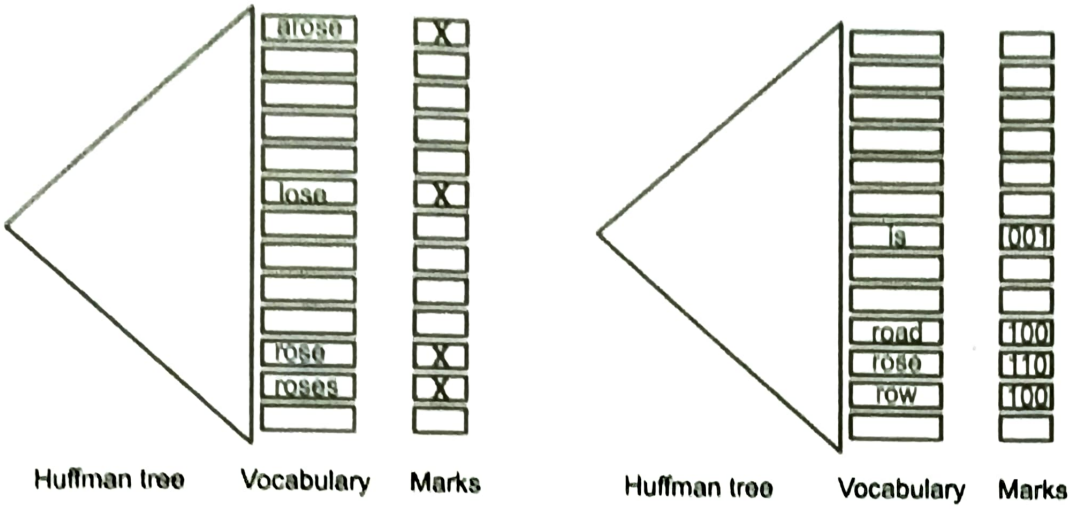
- Compression is crucial when storage space is limited, as it is when indexing the Web.
- Traditional thinking held that compression and searching were mutually exclusive operations.

- Texts that weren't going to be searched could be compressed, but you had to decompress a text before you could search it.
- In recent years, highly effective compression methods have emerged that enable direct text searching.
- Additionally, since the CPU times are comparable but the disc times are far shorter, the search performance is enhanced. In the end, everyone benefits from this.

☞ Sequential Searching

- In fact, one of the most effective methods for directly searching compressed text uses Huffman coding, which treats words as symbols.
- In other words, use each distinct text word as a symbol, tabulate their frequency distribution, and create a Huffman code for the words.
- After that, compress the text by inserting its code in place of each word.
- The Huffman code substitutes bytes for bits in its alphabet to increase compression/decompression efficiency.
- The entire text's vocabulary, or the list of all text terms, must be stored since Huffman coding requires to save the codes for each symbol. In order to efficiently search complex queries, this is fully exploited.
- Any single-word or pattern search begins with a vocabulary search. Some queries can be binary searched, whilst others, like approximate or regular expression searches, need a sequential search across the entire vocabulary.
- The list of several terms that correspond to the query is obtained once the search is finished. All those words' Huffman codes are compiled, and the compressed text is searched using them.
- One method is to simultaneously move through the Huffman decoding tree and the compressed text bytes, so that each time a leaf is reached, it is checked to see if it has been marked as "matching" the query or not. Fig. 5.7.1 gives an illustration of this. The search process can be sped up by using Boyer-Moore filtering.
- Phrase problems are a little trickier to solve. The vocabulary is searched for each component.
- Each word in the lexicon has a unique bit mask defined for it.

- When a word matches the ith element of the phrase query, we set the i-th bit in the mask of all matching words. Along with the Shift-Or algorithm, this is employed.



(1E19)**Fig. 5.7.1 : On the left, searching for the simple pattern 'rose' allowing one error. On the right, searching for the phrase 'ro* rose is,' whre 'ro*' represents a prefix search**

- When it comes to processing a complex question, which would take time with a typical algorithm, this technique is extremely quick. This is so that the algorithm can mostly ignore the intricacy of the first question as the complex search is only performed within the small vocabulary.
- Its CPU times are a little bit longer than Agrep's for a simple pattern. However, compressed searching outperforms all online algorithms when I/O times are taken into account.
- This strategy is by far the best for complex queries. When using the speedup technique, simple searches can increase to 18 Mb/sec from the reference machine's 14 Mb/sec CPU timings.

☞ Compressed Indices

Inverted Files

- Compression works quite well with inverted files. This is due to the fact that the lists of occurrences are arranged in ascending text positional order.
- As a result, it is clear to choose to illustrate the differences between the prior and current positions.

- By utilising methods that favour small numbers, these discrepancies can be represented with less space. You'll see that the discrepancies are smaller for longer lists.
- It's crucial to note that performance in terms of time does not always suffer from compression.
- The disk transfer takes up the majority of the time while responding to a query.
- Maintaining the index's compression allows for the delivery of less data, which may make the CPU work required to decompress it worthwhile.
- Keep in mind that a differential compression has no effect on the typical sequential traversal of the lists of occurrences.
- According to reports, query times for compressed and decompressed indices are comparable.
- Additionally, the text can be compressed separately from the index. Only the text necessary for display or, in the case of block addressing, for traversal will be decompressed.
- It is feasible to combine the two methods (compression and indexing) so that the vocabulary used for each activity is the same and the text is not decompressed before being indexed or searched.

☞ Suffix Trees and Suffix Arrays

- It is exceedingly challenging to further compress suffix arrays. This is so that they can represent a nearly random permutation of the text points.
- However, research has been done on the topic of creating suffix arrays from compressed text.
- The primary benefit is that both index building and querying almost quadruple their performance, which offsets the fact that the index and compressed text take up less space than the uncompressed text.
- Because fewer text blocks are required and more compressed text may fit in the same amount of memory, construction is completed more quickly.
- Because disc seek operations over the text area to compare suffixes take up a significant portion of the search time, searching is faster.
- The seeks are proportionally reduced as the text gets smaller.
- A Hu-Tucker coding is used in place of the Huffman code for words.

- Direct binary search over the compressed text is possible because the Hu-Tucker code respects the lexicographical relationships between the words (this is necessary at construction and search time).
- A very tiny percentage (2–3% in practise, with an analytical top constraint of 5%) makes this code unsatisfactory.
- If compression is applied, the reference machine's indexing rates for 250 Mb of text are close to 1.6 Mb/min, while query times are halved to 0.5 seconds overall and 0.3 seconds for the text alone.
- The overall search time should be reduced to 0.15 seconds by supra-indices.

☞ Signature Files

- Alternative techniques for compressing signature files abound. The common factor among all of them is that only few bits throughout the entire file are set.
- The opportunity arises to encode the bits that are not set using effective techniques, like run-length encoding.
- Different issues come into play depending on whether the file is kept as a series of bit masks or with one file for each bit in the mask.
- They enable us to either enhance B (to decrease the false drop probability) while maintaining the same space overhead, or they enable us to decrease space and hence disc times.
- There are reports of compression ratios around 70%.

⏭ 5.8 MULTIMEDIA IR : INDEXING AND SEARCHING : A GENERIC MULTIMEDIA INDEXING APPROACH

GQ. Define Indexing? What is Multimedia IR?
GQ. Differentiate process of information extraction and the process of document indexing?
GQ. Explain GEMINI algorithm in detail?
GQ. Describe Generic Multimedia Indexing Approach?

- In a variety of application scenarios, including offices, CAD/CAM applications, and medical applications, the requirement for an integrated management for multimedia data is expanding rapidly.

- Multimedia information systems are one of the most promising subfields in the field of information management due to this.
- The diversity of data that a multimedia information system must be able to support is its most crucial feature.
- Multimedia systems need to be able to store, retrieve, transport, and display data with a wide variety of characteristics, including text, still and moving images, graphs, and sound.
- Because of this, creating a multimedia system is much more difficult than creating a conventional information system.
- Traditional systems only work with straightforward data types like texts and numbers.
- On the other hand, a multimedia system's access and storage techniques, query language, and underlying data model must be able to accommodate objects with a very complicated structure.
- Because many multimedia applications need to organize their data, at least in part, multimedia IR systems necessitate some kind of database schema.
- To enable greater flexibility in data organization, it might be necessary to weaken the notion of schema relative to the traditional notion.
- A Multimedia IR system must also handle metadata, which is essential for data retrieval but not required by standard IR systems.
- The particular qualities of multimedia data and the types of operations to be carried out on such data are the two key determinants of the architecture of a Multimedia IR system. The sections that follow briefly discuss both of these elements.

Definition : Given two objects, O_1 and O_2 the distance (= dissimilarity) of the two objects in denoted by

$$D(O_1, O_2) \qquad ...(5.8.1)$$

For instance, the distance P() might be their Euclidean distance if the objects are two (equal-length) time series (the root of the sum of squared differences).

Similarity searches can be divided into two groups :

(1) Whole match

- We are looking for data items that are around Q, given a set of N objects O_1, O_2,.., On and a query object Q.
- Be aware that the query and the objects are of the same type; for instance, if the objects are 512 × 512 grayscale images, the query is also 512 × 512 grayscale images.

(2) Sub-pattern match

- In this case, the query can only specify a portion of the object. In more detail, given N data objects (for example, pictures), O_1, O_2,.
- We are looking to discover the components of the data objects that fit the query based on a query (sub-) object Q and a tolerance e.
- The query in this situation could be, for example, a 16 × 16 sub pattern if the objects are, for example, 512 × 512 grayscale images (like medical X-rays) (e.g., a typical X-ray of a tumor).

(3) A Generic Multimedia Indexing Approach

We'll concentrate on "whole match" searches to show the fundamental concept. The issue for these queries is defined as follows :

- There are N objects in our collection. O_1, O_2, ... O_n
- The function $D(O_i, O_j)$ provides the distance/dissimilarity between two objects (O_i, O_j), and it can be implemented as a (potentially slow) programme.
- The user enters a tolerance e and a query object Q.

Our objective is to locate the collection items that are closest to the query object by distance e. Applying sequential scanning is a clear-cut answer. For each object, O_i (1<=i<=n), we may calculate its distance from Q and report the objects with distance $D(Q, O_i) = e$.

Sequential scanning, however, could be slow for two reasons :

(1) The cost of computing the distance may be high.

(2) The size N of the database may be enormous.

So, we're seeking for a quicker substitute. Next, we'll discuss the

GEMINI (GEneric Multimedia object INdexIng) approach, which is built on two concepts that each aim to mitigate one of sequential scanning's two drawbacks :

- The use of spatial access methods to facilitate faster-than-sequential searching;
- A "quick-and-dirty" test to swiftly eliminate the large majority of non-qualifying objects (perhaps allowing some false alarms).

Consider a database of time series, such as yearly stock price movements, with one price per day. Assume that the distance function between two such series *S* and Q, is the Euclidean distance

$$D(S, Q) \equiv \left(\sum_{i=1} (S[i] - Q[i]^2) \right)^{1/2} \quad ...(5.8.2)$$

where *S(i)* stands for the value of stock *S* on the i-th day. Clearly, computing the distance of two stocks will take 365 subtractions and 365 squarings in our example.

- The quick-and-dirty test's goal is to characterize each sequence with a single number in order to eliminate as many non-qualifying sequences as possible.
- An example of such a figure would be the average stock price for the year.
- It is obviously impossible for two stocks to be similar if their average prices significantly diverge from one another.
- We may experience false alarms because the contrary is not true. We'll refer to numbers that provide some context for a sequence as features.
- With one numerical comparison for each sequence, we may quickly test a set of stocks by using a good characteristic (like the "average" in the stock prices example).
- We can ultimately map each of our things into a point in f-dimensional space by applying f characteristics to each of our objects. This mapping will be referred to as F() (for 'F'eature) :

Definition Let F() be the mapping of objects to f -dimensional points, that is, F(0) will be the f -D point that corresponds to object O.

- The solution to the second problem with sequential scanning is found in this mapping : by putting these f-D points into a spatial access technique, we may group them into a hierarchical structure similar to the R*-trees.
- We can use the R*-tree to remove sizable chunks of the database that are uninteresting after a query.
- Therefore, we don't even need to perform the fast test on each of the f-D points.
- The fundamental concept is depicted in Fig. 5.8.1 : 2D points are used to map objects. A f-D sphere in feature space, centered on the image F(Sn) of Sn, results from the "whole match" query, which demands all the objects that are comparable to Sn within tolerance e.
- Exactly these queries on multidimensional points are what R*-trees and other SAMs are made to efficiently respond to. The search algorithm for a whole match query is as follows in more detail:

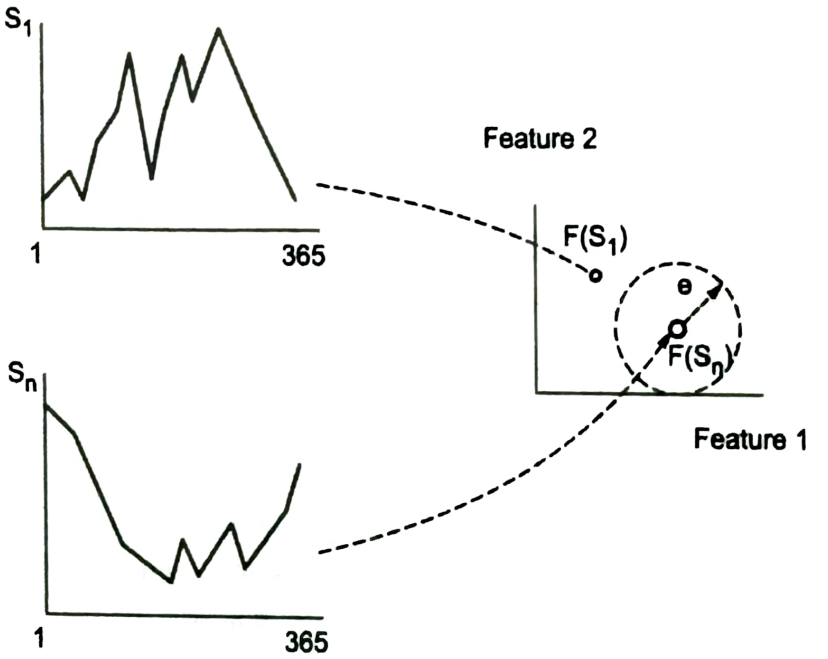


Fig. 5.8.1 : Illustration of the basic idea: a database of sequences *S1*, ..., *Sn* each sequence is mapped to a point in feature space; a query with tolerance e becomes a sphere of radius e.

Algorithm 1 *Search :*

(1) Map the query object Q into a point F(Q) in feature space.

(2) Using a spatial access method, retrieve all points within the desired tolerance.

(3) Retrieve the corresponding objects, compute their actual distance from Q and discard the false alarms.

- It would seem that the strategy could solve both issues with the sequential scan, leading to significantly quicker searches.
- The only step that requires caution is ensuring that the distances are not distorted by the mapping F() from objects to f-D points. Let $D_{feature}()$ be the (say, Euclidean) distance of the relevant feature vectors and D() be the distance function between two objects.
- In a perfect world, the mapping would maintain the distances precisely, in which case the SAM wouldn't experience false alarms or dismissals.
- It might be challenging to demand exact distance preservation, though.
- The significant observation is that if the distance in feature space matches or underestimates the distance between two objects, we can be sure that there won't be any incorrect dismissals.
- This suggests that things should appear closer using our mapping F() from objects to points.
- Let O_1 and O_2 be two objects (for example, identical sequences) with the distance function D() (for example, the Euclidean distance) and $F(O_1)$ and $F(O_2)$ being their feature vectors (for example, their first few Fourier coefficients) with the distance function $D_{feature}()$. Next, we have:
- **Lemma 5.1 (Lower Bounding)** To guarantee no false dismissals for whole- match queries, the feature extraction function F() should satisfy the following formula:

$$D_{feature}(F(O_1), F(O_2)) \leq D(O_1, O_2) \quad ...(5.8.3)$$

- For range queries, lower-bounding the distance is effective. Will it function properly for the other relevant queries, such as "all pairs" and "nearest neighbor" ones? Both questions have an affirmative response.

Algorithm 2 (GEMINI) **GE**neric **M**ultimedia object **IN**dex**I**ng approach :

(1) Determine the distance function D() between two objects.

(2) Find one or more numerical feature-extraction functions, to provide a 'quick-end-dirty' test.

(3) Prove that the distance in feature space lower-bounds the actual distance D(), to guarantee correctness.

(4) Use a SAM (e.g., an R-tree), to store and retrieve the f-D feature vectors.

- The first two GEMINI steps merit further discussion. A domain specialist is involved in the initial step.
- The methodology only focuses on search speed; the expert's distance function is completely responsible for the accuracy of the results. Thus, GEMINI will only be faster than a sequential database scan while still returning the same set of responses that would have been returned.
- GEMINI's second step involves both imagination and intuition. It begins by attempting to respond to the query:
- "Feature-extracting" What numerical feature should be used to characterize each data object if we are only allowed to utilize one?

5.9 AUTOMATIC FEATURE EXTRACTION

GQ. Discuss the process of information extraction and the process of document indexing?

- Any environment from which we may extract features is useful for GEMINI. There are, in fact, automatic feature extraction algorithms like "Multidimensional Scaling" (MDS) and "FastMap."
- We can plot a 2D or 3D projection of the data set and look for clusters, correlations, and other patterns after extracting features. This not only makes it easier to employ pre-made spatial access methods.
- Fig. 5.9.1 displays the FastMap findings on 35 documents from seven classes after k = 3 features/dimensions were derived. The courses cover topics like cookery techniques ('Rec'), abstracts of computer science technical reports ('Abs'), and basketball reports ('Bbr').
- The cosine similarity function was a decreasing function, and so was the distance function.
- The 3D scatter-plot is displayed in the picture both in its whole and after zooming in on the center to demonstrate how well FastMap can cluster data. Just take note of how effectively the seven classes are divided into simply k = 3 dimensions.

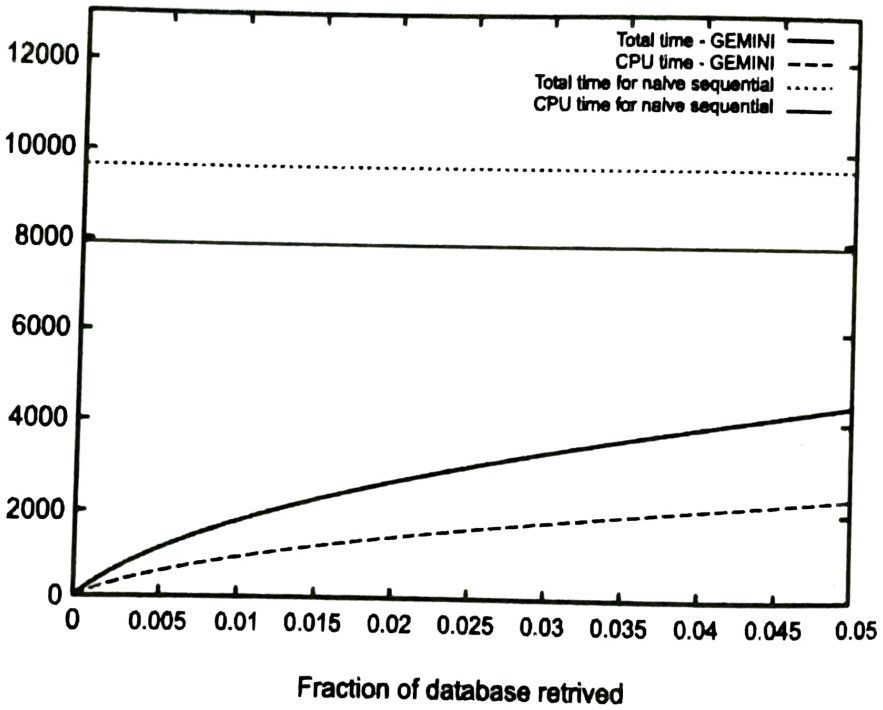


Fig. 5.9.1 : Response time vs. selectivity, for the sequential ('naive') retrieval and for GEMINI.

5.10 SEARCHING THE WEB

- There are essentially three ways to search the Web. Two of them are famous and regularly employed.
- Use of search engines that index a subset of Web content as a full-text database is the first option. Utilizing Web directories, which group particular Web documents by subject, is the second option.
- The third, which is still under development, is to use the Web's hyperlink structure to search it. Here, we discuss all three types of Web searches.
- We first talk about the difficulties of searching the Web, then we look at some Web statistics and models that can help us grasp how complex the issue is.
- The primary tools used to search the Web nowadays are then covered in detail. Search examples, Web directories, hybrid systems, user interfaces, and search engines are discussed.
- New query languages that take advantage of the Web's graphical nature are still being developed. Finally, we review recent developments and research concerns.

☞ Challenges

We now discuss the key issues the Internet causes. They fall into two categories: issues with the data itself and issues with the user and how he or she interacts with the retrieval system. The following are the issues with the data :

(1) Distributed data
(2) High percentage of volatile data
(3) Large volume
(4) Unstructured and redundant data
(5) Data quality
(6) Heterogeneous data

(1) Distributed data

- Because of the fundamental characteristics of the Web, data is spread across numerous systems and platforms.
- These machines are linked together without following a predetermined topology, and the stability and bandwidth of the network connections vary greatly.

(2) High percentage of volatile data

- New computers and data can be added or removed with ease thanks to Internet dynamics.
- When domain names or file names change or vanish, we also experience dangling links and relocation issues.

(3) Large volume

The Web's exponential expansion presents challenging scaling problems.

(4) Unstructured and redundant data

- The Web is referred to as a distributed hypertext by the majority of people. This is not exactly the case, though.
- Any hypertext has an underlying conceptual model that keeps the information ordered and consistent while also adding hyperlinks.

- Even for individual documents on the Web, it is hardly ever the case. Each HTML page is also poorly structured, and the term "semi-structured data" is sometimes used.
- Furthermore, a lot of the information on the Web is redundant or strikingly similar. Near duplicates make up 30% of all Web pages. There may be even more semantic redundancy.

(5) Data quality

- The Web can be thought of as a brand-new publication medium. There is typically no editorial procedure, though.
- As a result, data may be inaccurate, invalid (for instance, if it is too old), poorly worded, or frequently contain several errors from various sources.

(6) Heterogeneous data

Heterogeneous data in addition to dealing with various media types and consequently various formats, we also have various languages and, worse than, several alphabets, some of which are very huge (for example, Chinese or Japanese Kanji).

The majority of these issues cannot be resolved only by software upgrades. Many of them won't even change because they are issues that are inherent in human nature.

Therefore, the main task is to provide a strong search query to the search system and return a controllable and relevant answer, despite the inherent issues the Web poses. Furthermore, we should strive to accomplish the second objective in practise, even for poorly constructed queries.

☞ Characterizing the Web

Measuring the Web

- Due to its extreme dynamic nature, measuring the Internet, and particularly the Web, is a challenging endeavor.
- Today, there are more than 40 million computers connected to the Internet in more than 200 countries, many of which are Web servers.
- According to the report, there are between 2.4 million and over three million Web servers worldwide (October 1998).
- When we take into account that there are numerous Web sites that use

virtual hosts to share a single web server, that not all of them are fully accessible, that many of them are provisional, etc., this large range may be explained.

- The Web doubled in size in nine months between 1997 and 1998, and it now adds 20 million pages every month.
- The 30,000 largest Web sites, on the other hand, are thought to contain about 50% of all Web pages.
- Following HTML in popularity are GIF and JPG, ASCII text, Postscript, and then GIF and JPG again. GNU zip, Zip, and Compress are the most widely used compression programmes.
- A typical HTML page is what? First off, the majority of HTML pages are not standards, meaning that they do not adhere to all HTML requirements.
- Despite being an example of SGML, HTML documents hardly ever begin with a formal document type description.
- There have been three studies done on the languages used in web pages. Funredes conducted the initial investigation between 1996 and 1998.
- It makes use of the AltaVista search engine and operates by looking up various words in various languages.
- The results of this method are consistent with the second study, which was conducted by Alis Technology and is based on automatic software that can recognise the language used, even though it may not be statistically significant.
- The study's testing of such software was one of its objectives (done in 8000 Web servers). In June 1998, OCLC conducted its most recent investigation utilizing a sample of Internet numeric addresses and the SILC language identification programme.
- With the exception of the OCLC data, which counts websites, Table 5.10.1 lists the percentages of Web pages authored in each language together with the population (in millions) that speaks it.
- The differences for Japanese could be the result of a failure to recognise pages written in Kanji.
- Some languages, particularly Spanish and Portuguese, are expanding quickly and will soon overtake French. Over 100 different languages are spoken worldwide.

Table 5.10.1 : Languages of the Web

Language	Funredes (1998, %)	Alis Tech. (June 1997, %)	OCLC (June 1998, %)	Spoken by (millions)
English	76.4	82.3	71	450
Japanese	4.8	1.6	4	126
German	4.4	4.0	7	118
French	2.9	1.5	3	122
Spanish	2.6	1.1	3	266
Italian	1.5	0.8	1	63
Portuguese	0.8	0.7	2	175

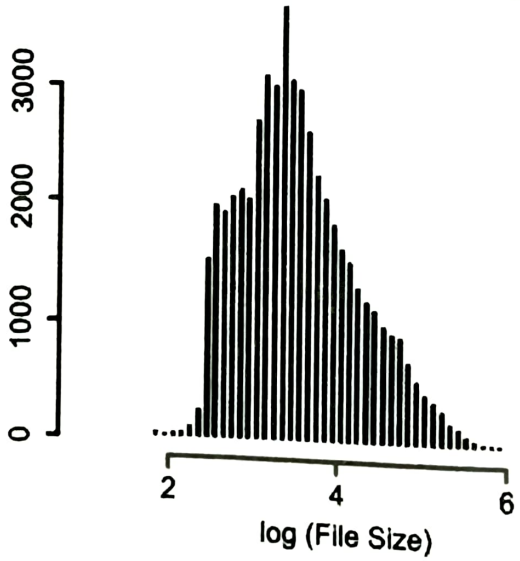
☞ Modeling the Web

- The distribution of document sizes is a subject of another model. The document sizes are self-similar, or have a wide range, in accordance with this concept. One of two distinct distributions can be used to model this.
- The distribution's main body follows a logarithmic normal distribution, and as a result, the probability of finding a document with a size of z bytes is given by

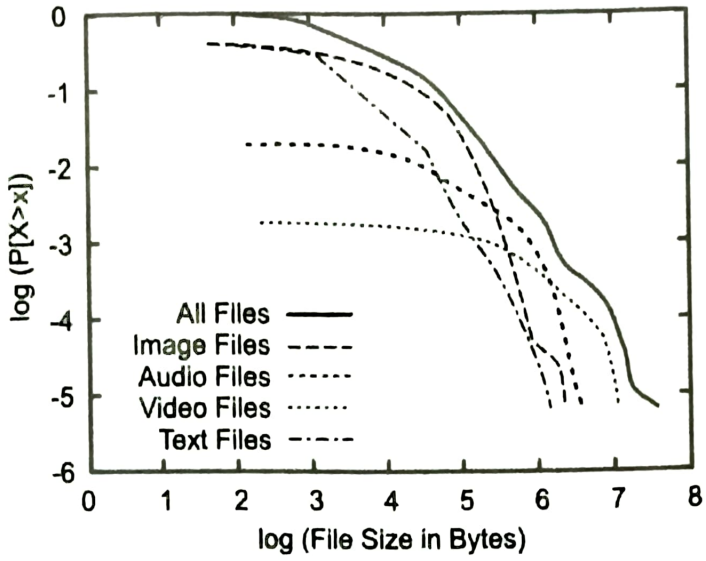
$$p(x) = \frac{1}{x\,\sigma\sqrt{2\pi}} \exp - (\ln x - \mu)^2 / 2\sigma^2$$

where the average (μ) and standard deviation (σ) are 9.357 and 1.318

- Fig. 5.10.1 (left) shows the size distribution of the experimental data.



(a)



(b)

Fig. 5.10.1 : Left : Distribution for all file sizes (courtesy of M. Crovella, 1998). Right: Right tail distribution for different file types (from Crovella and Bestavros, 1996). All logarithms are in base 10.

- The right tail of the distribution is 'heavy-tailed.' That is, the majority of documents are small, but there is a non-trivial number of large documents. This is intuitive for image or video files, but it is also true for HTML pages.

$$p(x) = \frac{\alpha k^{\alpha}}{x^{1+\alpha}}$$

where x is measured in bytes and k and α are parameters of the distribution

- α is approximately 1.36 for text files, while it is less for pictures and other binary forms. After accounting for all Web documents, we arrive at $\alpha = 1.1$ and $k = 9.3$ Kb.
- In other words, the cut point between the two distributions is 9.3 Kb, and 93% of all the files are smaller than this threshold.
- In actuality, photos make up the majority of files under 50 kilobytes, audio files become more prevalent between 50 and 300 kilobytes, and video files become more prevalent above that, up to several megabytes.
- The parameters for these distributions were derived from a sample of more than 54,000 Web pages that were requested over the course of two months in 1995 by a variety of users.

5.11 SEARCH ENGINES

GQ. Define Crawling.
GQ. What is a search engine. Discuss the architectures of Search engine?
GQ. Explain Ranking with an example?

- Standard IR systems and the Web vary primarily in that all queries on the Web must be resolved without referencing the content (that is, only the indices are available).
- Otherwise, that would necessitate either keeping a copy of the Web pages locally (expensive) or using the network to get remote pages at query time (too slow). This distinction affects the indexing and searching algorithms as well as the available query languages.

Architecture

- A centralised crawler indexer design is used by most search engines.
- Programs called crawlers (also known as software agents) scour the Web and submit fresh or updated pages to a central server where they are indexed.
- Robots, spiders, wanderers, walkers, and know bots are other names for crawlers. Contrary to what its name would suggest, crawlers don't actually move to and operate on distant machines; instead, they run on a local system and make queries to distant Web servers.
- In a centralized manner, the index is utilised to respond to inquiries coming from various Web locations. A search engine's software design based on the AltaVista architecture is shown in Fig. 5.11.1.

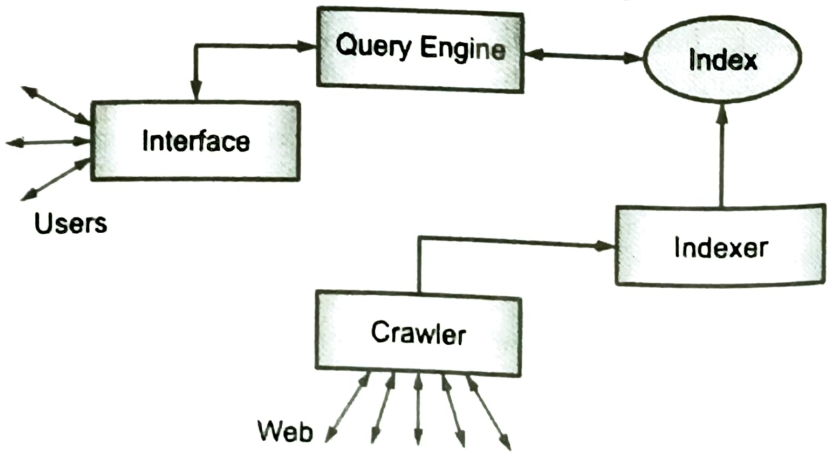


Fig. 5.11.1 : Typical crawler-indexer architecture

- It is divided into two sections: one that works with people and includes a user interface and a query engine, and another that includes crawler and indexer modules.
- Twenty multi-processor machines with a combined capacity of more than 130 Gb of RAM and more than 500 Gb of disc space were used to run the entire AltaVista system in 1998. More than 75% of these resources are only used by the query engine.
- Due to the very dynamic nature of the Web, the congested communication channels, and the heavy strain on Web servers, the primary issue this architecture faces is the collection of data. The amount of data is a serious issue as well.
- In fact, in the near future, Web growth may be too much for the crawler-indexer design to handle. Good load balancing is crucial between a search engine's various internal (answering questions and indexing) and exterior (searching) activities (crawling).
- The majority of search engines are American-based and concentrate on English-language documents. However, there are search engines that are focused on certain regions or languages that may, for example, query and retrieve documents written in Kanji (Chinese, Japanese, and Korean).
- There are search engines that adopt different strategies as well. For example, Ask Jeeves! simulates an interview, while Direct Hit lists the Web pages in the response in order of popularity.
- We should also add search engines that are focused on particular themes, such DejaNews, which searches USENET archives, and the Search Broker , which allows us to search in a variety of specific topics.
- Additionally, there are engines to retrieve certain Web sites, such as home pages for individuals or organizations, or particular items, such as photos, software applets, or electronic mail addresses.

☞ Distributed Architecture

- The crawler indexer architecture comes in a variety of forms. The most crucial of them all is Harvest.
- Harvest uses a distributed architecture, which is more effective than the crawler architecture, to collect and distribute data.

- The biggest disadvantage of Harvest is that it necessitates the coordination of numerous Web servers.
- The Harvest distributed approach addresses a number of issues with the crawler-indexer architecture, including :
  (1) Web servers receive requests from various crawlers, increasing their workload;
  (2) Web traffic increases because crawlers retrieve entire objects, but most of their content is ignored; and
  (3) Information is gathered independently by each crawler, without coordination between all of the search engines.
- Harvest provides two key components gatherers and brokers to address these issues. A gatherer pulls indexing data from one or more Web servers and extracts it.
- The mechanism predetermines recurring gathering times. The indexing system and query interface for the acquired data are provided by a broker.
- Brokers update their indexes incrementally by retrieving data from one or more gatherers or from other brokers.
- Server load and network traffic can be improved in different ways depending on how gatherers and brokers are set up. For instance, a gatherer may be installed on a web server and prevent any outside traffic from reaching it.
- Additionally, a gatherer can deliver data to multiple brokers, cutting down on repetitive work.
- Information can be filtered by brokers and sent to other brokers. This layout enables the flexible and general sharing of labor and information. Fig. 5.11.2 depicts a Harvest architecture illustration.
- Replicators and object caches are further features of the Harvest design. Scalability of user bases can be improved by replicating servers using a replicator.
- To enable quicker access, the registration broker, for instance, can be replicated in various geographical locations.
- Replication can be used to distribute the collection process across numerous Web servers.

- The object cache also lessens the stress on servers and the network as well as the response time for Web page requests.

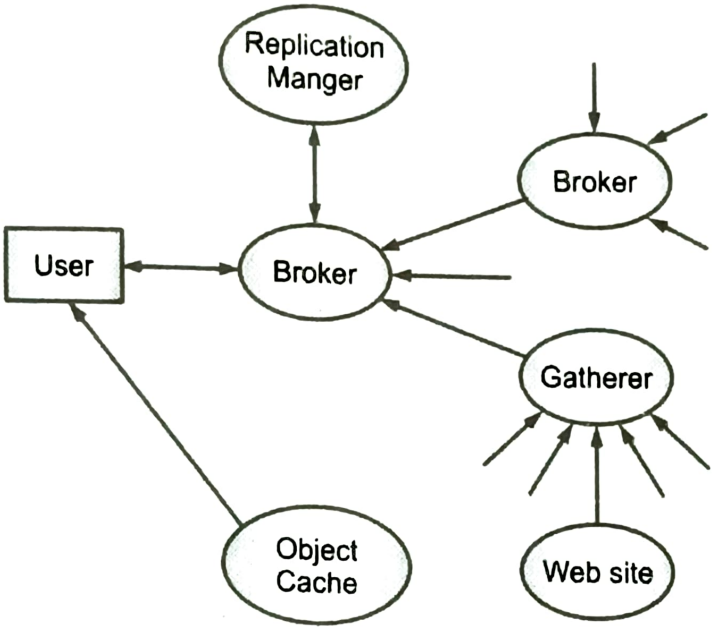


Fig. 5.11.2 : Harvest Architecture

☞ User Interfaces

- The query interface and the answer interface are two crucial components of search engines' user interfaces.
- A box where one or more words can be written serves as the basic query interface.
- Although a user might anticipate it, a particular string of words does not always correspond to the same query across all search engines. For instance, a word sequence in AltaVista refers to the collection of all Web pages with at least one of those words, whereas in HotBot it refers to Web pages with all of the words.
- Another issue is that search engines vary in their usage of stopwords, stemming, and case sensitivity, making it impossible to determine the text's logical order.
- In addition, each search engine has a query interface for more complicated searches, a command language with Boolean operators, and additional capabilities like phrase search, proximity search, and wild cards.

- The complicated query interfaces for the three top search engines offer a number of filtering options. Language, country or Internet domain, date range, or the presence of particular data types, such as images or audio, can be used to filter the results.
- Additional words that must be present or absent from the answer or in a specific field, such as the URL or title, can also be used to filter the results.
- The solution often includes a list of the top ten Web pages.
- This list's entries each contain some details about the document they each refer to.
- The information typically comprises the URL, size, the date the page was indexed, and a few lines describing its content (title plus first lines or selected headings or sentences).
- Though this is sometimes fixed or limited to a few options, some search engines let users modify the number of pages returned in the list and the amount of information each page.

☞ Ranking

- The majority of search engines perform ranking using variants of the Boolean or vector paradigm.
- Ranking must be done without using the text; only the index, same like when searching.
- The particular ranking algorithms that the modern search engines use are not well known. Furthermore, given their variances and ongoing developments, it is challenging to compare fairly distinct search engines.
- More importantly, it is very impossible to assess recall because there may be several relevant pages for straightforward questions.
- In addition to the traditional tf-idf approach, Yuwono and Lee present three additional ranking algorithms. Boolean spread, vector spread, and most-cited are their names.
- The first two are extensions of the standard Boolean and vector model ranking algorithms that take into account pages that are pointed to by a page in the answer or that point to a page in the answer.
- The third, which receives the most citations, is only supported by the terms found on pages that connect to the pages in the answer.
- The vector model produces a better recall-precision curve, with an average precision of 75%, according to a comparison of these two

methods using 56 queries across a sample of 2400 Web pages.

- Additionally, some of the new ranking algorithms make advantage of hyperlink data. This is a crucial distinction between traditional IR databases and the Web.

☞ Crawling the Web

- As there are various methods, we go over how to crawl the web in this part. The easiest method is to begin with a set of URLs and then, either in a breadth-first or depth-first approach, extract further URLs from that set.
- Search engines enable users to add top Web sites that will be included to the URL set, for this purpose. Starting with a collection of well-known URLs is an alternative, as we can assume that they include information that is regularly sought.
- Both scenarios work well for a single crawler, but it is challenging to coordinate multiple trawlers to prevent repeatedly accessing the same page.
- Another method is to divide the Web into segments using Internet addresses or country codes, assign one or more robots to each segment, and thoroughly examine each segment.
- The index of a search engine can be compared to the stars in the night sky when considering how the Web is browsed.
- Since the light had to travel over various distances in order to reach our sight, what we perceive has never existed. Similar to this, Web pages mentioned in an index were looked up at various times and might not be accessible now.
- Nevertheless, we acquire a page's true content when we retrieve it. How recently updated are the Web pages listed in an index?
- The age of the pages ranges from one day to two months. Because of this, the date that the page was indexed is typically displayed in the search engine's response.
- There are between 2 and 9% invalid links that are retained by search engines.
- It's crucial to follow the URLs in the correct order. The links on a Web page can be followed either width first or depth first, as was already explained. By employing a breadth-first approach, we start by looking at every page that is connected from the current page, and so on.

- This fits very well with websites that are organised according to relevant topics.
- In contrast, the coverage will be broad but shallow, and a web server may receive a deluge of quick requests.
- In the depth-first instance, we recursively return after clicking the first link on a page and continuing until we can no longer go any farther. This offers a deep but slender traverse.

5.12 BROWSING

GQ. Explain in short Web based browsing?

- This section discusses web-based browsing and search tools, particularly Web directories.
- The Web coverage provided by directories is very low (less than 1% of all Web pages), the answers returned to the user are usually much more relevant.

Web Directories

- The most popular search engine, Yahoo!, is the greatest and oldest example of a Web directory.
- eBLAST, LookSmart, Magellan, and NewHoo are a few other significant online directories.
- Some of them are hybrids since they offer searches over the entire Web. These days, the majority of search engines offer topic categories as well. Examples include AltaVista Categories, AOL Netfind, Excite Channels, HotBot, Infoseek, Lycos Subjects, and WebCrawler Select.
- There are websites devoted to business, news, and in particular study bibliography, as an example.
- Yellow pages, subject directories, and catalogues are other names for web directories. The URLs of the most significant Web directories are provided in Table 5.12.1.

Table 5.12.1 : URLs, Web pages indexed and categories (both in thousands) of some Web directories (beginning of 1998).

Web directory	URL	Web sites	Categories
eBLAST	www.eblast.com	125	–
LookSmart	www.looksmart.com	300	24
Lycos Subjects	a2z.lycos.com	50	–
Magellan	www.mckinley.com	60	–
NewHoo	www.newhoo.com	100	23
Netscape	www.netscape.com	–	–
Search.com	www.search.com	–	–
Snap	www.snap.com	–	–
Yahoo!	www.yahoo.com	750	–

Table 5.12.2 : The first level categories in Web directories

Arts and Humanities	Local
Automotive	News
Business and Economy	Oddities
Computers and Internet	People
Education	Philosophy and Religion
Employment	Politics
Entertainment & Leisure	Recreation
Games	Reference
Government	Regional
Health and Fitness	Science and Technology
Hobbies and Interests	Shopping and Services
Home	Social Science
Investing	Society and Culture
Kids and Family	Sports
Life and Style	Travel and Tourism
Living	World

- Directories are taxonomies that classify human knowledge in a hierarchical manner.
- The first level of the taxonomies utilised by Web directories is displayed in Table 5.12.2. (the number of first level categories ranges from 12 to 26). There are a few subcategories available on Web directories' home pages as well, bringing the total number of topics to about 70.

5.13 METASEARCHERS

- A particular query is sent to various search engines, directories, and other databases by metasearchers, which then gather and combine the results.
- Metacrawler and SavvySearch are two examples. The capacity to integrate results from multiple sources and the capability for users to submit the same query to multiple sources through a single standard interface are the key benefits of metasearchers.
- The way that ranking is carried out in the unified result (in some situations, no ranking is done) and how successfully the user query is translated to the unique query language of each search engine or Web directory are two ways that metasearchers differ from one another.
- Table 5.13.1 shows the URLs of the main metasearch engines as well as the number of search engines, Web directories and other databases that they search.

Table 5.13.1 : URLs of metasearchers and number of sources that they use (October 1998).

Metasearcher	URL	Sources used
Cyber 411	www.cyber411.com	14
Dogpile	www.dogpile.com	25
Highway61	www.highway61.com	5
Inference Find	www.infind.com	6
Mamma	www.mamma.com	7
MetaCrawler	www.metacrawler.com	7

Metasearcher	URL	Sources used
MetaFind	www.metafind.com	7
MetaMiner	www.miner.uol.com.br	13
MetaSearch	www.metasearch.com	-
SavvySearch	savvy.cs.colostate.edu:2000	>13

- The benefits of metasearchers include the ability to arrange the results by several criteria, such as host, keyword, date, etc.; this can make the results more insightful than those from a single search engine.
- Consequently, browsing the results ought to be easier. However, because the metasearcher can only retrieve a certain amount of results from each search engine, the result may not always include all the Web pages that fit the query (it can be changed by the user, but there is an upper limit).
- Nevertheless, the pages that many search engines return ought to be more pertinent.

5.14 SEARCHING USING HYPERLINKS

GQ. What are Software agents?
GQ. Write a short note on Web Query languages?

In this part, we discuss additional paradigms for searching the Web that rely on its hyperlinks. Dynamic searching and Web query languages are some of them. Due to a number of factors, including performance restrictions and a lack of commercial goods, these concepts are still not extensively adopted.

☞ Web Query Languages

- The content of each page has been the basis for questions up until this point. However, inquiries might also contain the web page link architecture. For instance, we'd want to find all the Web pages that have at least one image and can be accessed from a specific site by following no more than three links.
- It has been possible to ask this kind of question by using various data models. The most crucial ones are a semi-structured data model to represent the content of Web sites and a labeled graph model to represent

Web pages (nodes) and hyperlinks (edges) between Web pages.

- In the latter type, the data structure is typically unknown, subject to evolution, and potentially expansive and descriptive.
- The original generation of Web query languages were focused on merging content with structure, despite the fact that several models and languages for querying hypertext had already been proposed before the Web's inception.
- These languages mix graph queries that describe link structure with patterns found in the documents (using path regular expressions). W3QL, WebSQL, WebLog, and WQL are some of them.
- The emphasis on semi-structured data is maintained in the second generation of languages, referred to as Web data manipulation languages.
- The original generation of Web query languages were focused on merging content with structure, despite the fact that several models and languages for querying hypertext had already been proposed before the Web's inception.
- These languages mix graph queries that describe link structure with patterns found in the documents (using path regular expressions).
- W3QL, WebSQL, WebLog, and WQL are some of them. The emphasis on semi-structured data is maintained in the second generation of languages, referred to as Web data manipulation languages.

☞ Dynamic Search and Software Agents

- Sequential text searching is the counterpart of dynamic search on the Web. The concept is to conduct an online search and click on links to find pertinent information.
- The key benefit is that you are searching in the current Web structure rather than what is contained in a search engine's index. Although this method is sluggish for the entire Web, it may be employed in compact and dynamic Web subsets.
- The fish search was the first heuristic to be developed, and it takes advantage of the common sense insight that relevant papers frequently have relevant neighbors.

- As a result, the search is directed by clicking on links in pertinent papers. Shark search, which evaluates nearby pages' relevancy more thoroughly, helped to enhance this.
- These algorithms' main goal is to prioritize links and follow them starting from a single page and a certain query. The page with the highest importance is examined at each stage.
- A heuristic decides whether or not to follow the links on that page if it is determined to be relevant. If so, new pages are placed in the proper places on the priority list.

General Questions

Q. 1 List out the differences between Boyer –Moore text search algorithm and Kunth-Pratt Morris algorithm?

Q. 2 Describe various measures used in the information system evaluation ?

Q. 3 Describe the process of creating inverted index with example

Q. 4 Compare Boolean queries & Natural Language queries ?

Q. 5 Differentiate process of information extraction and the process of document indexing ?

Q. 6 Define text search algorithm ?

Q. 7 Describe about the techniques involved in software text search algorithm ?

Q. 8 Compare statistical indexing and concept indexing ?

Q. 9 Discuss the different classes of automatic indexing ?

Q. 10 What are the tradeoffs between using the Aho-Corasick versus Boyer –Moore algorithms?

Q. 11 Discuss the process of information extraction and the process of document indexing ?

Q. 12 Explain which circumstances not required for manual indexing, to ensure finding information ?

Q. 13 Differentiate process of information extraction and the process of document indexing ?

Chapter Ends...

❑❑❑