Chapter 1.14
Indexing Textual Information

Ioannis N. Kouris
University of Patras, Greece

Christos Makris
University of Patras, Greece

Evangelos Theodoridis
University of Patras, Greece

Athanasios Tsakalidis
University of Patras, Greece

INTRODUCTION

Information retrieval is the computational discipline that deals with the efficient representation, organization, and access to information objects that represent natural language texts (Baeza-Yates, & Ribeiro-Neto, 1999; Salton & McGill, 1983; Witten, Moûat, & Bell, 1999). A crucial subproblem in the information retrieval area is the design and implementation of efficient data structures and algorithms for indexing and searching information objects that are vaguely described. In this article, we are going to present the latest developments in the indexing area by giving special emphasis to: data structures and algorithmic techniques for string manipulation, space efficient implementations, and compression techniques for efficient storage of information objects.

The aforementioned problems appear in a series of applications as digital libraries, molecular sequence databases (DNA sequences, protein databases [Gusûeld, 1997)], implementation of Web search engines, web mining and information filtering.

BACKGROUND

Dictionary Data Structures

The dictionary data structure stores a set S of n elements in order to support the operations of insertion, deletion, and the test of membership. A basic criterion for categorizing dictionary data structures is whether only comparisons are used, or the representation of elements for guiding the search is also employed. Typical representatives of the former group are *search trees* and of the latter *tries* and *hashing*. Search trees need O(log$n$) update/search time and O(n) space and the most prominent examples of them are: AVL-trees, red-black trees, (α,b)-trees, BB[α]-trees and Weight Balanced B-trees (Arge, & Vitter, 1996; Cormen, Leiserson, & Rivest, 1990; Mehlhorn, 1984). On the other hand, *tries* and *hashing* structures (Cormen, Leiserson, & Rivest, 1990; Czegh, Havas, & Majewski, 1997; Pagh, 2002) try to use the representation (for example, the value of the element written as a string of digits or the value itself), to compute directly the element's position in system's memory. The time and space complexities of these structures generally vary; however, it should be mentioned that a lately developed structure (Anderson & Thorup, 2001) answers both search and update operations in O ($\sqrt{\log n / \log\log n}$) time. This structure is also able to retrieve the largest element in the stored set smaller than a *query* element (*predecessor* query).

Finding Occurrences of Patterns

The string matching (or pattern matching) problem is one of the most frequently encountered and studied problems in the area of system/algorithm design. In this problem, we are searching for the occurrences of a pattern P in a sequence of symbols T. The naive $O(|P||T|)$ algorithm aligns the pattern at each one of the O(|T|) possible positions of the sequence and executes O(|P|) comparisons; however, there exist elegant, though complex, algorithms whose overall time complexity is linear, that is, $O(|P|+|T|)$. The most known linear time algorithms that achieve that are Knuth-Morris-Pratt (Knuth, Morris, & Pratt, 1977) and variants of the Boyer-Moore (Boyer & Moore, 1977) algorithm. For the case that we are searching for a set of patterns in the sequence, the Aho-Corasick automaton (Aho & Corasick, 1975) can be used. This automaton accepts all the patterns of the set and can be constructed in time linear to the sum of the lengths of them. Running the automaton with the characters of T, all the occurrences of the patterns are reported in $O(|T|)$ time.

On most of the modern applications, the patterns arrive in an on line manner and the $O(|T|+|P|)$ computational time is prohibitive; there is need for indexing structures (indices) that can perform the queries as closer as possible to $O(|P|)$ computational time, assuming that the text has been preprocessed *once*. The indices that try to satisfy this demand are divided in two categories: the *word-based* (or *keyword-based*) indices, which have been designed for sequences of symbols that can be divided in tokens/words, and the *full-text* (or *sequential scan*) indices, where the previous feature does not hold and the strings involved are non-tokenizable.

TEXT AND STRING DATA STRUCTURES

Word Based Indices

The most commonly used indexing structures in this category are *inverted files, signature files* and *bitmaps*. An inverted file consists of two parts: a structure for storing the set of all different words in the text and, for each such word, a list of the text positions where the word appearances are stored. Signature files are term-oriented structured based on hashing while bitmaps represent each document as a bit vector having length equal to the size of the lexicon. In typical applications compressed inverted files are considered to be superior to both signature files and bitmaps (Faloutsos, 1985; Zobel, Moffat, & Ramamohanarao, 1998).

More analytically, consider a document collection and a lexicon containing the terms that appear in the documents of the collection. An inverted file consists of a search structure containing all the distinct terms that appear in the lexicon and,