

New Indices for Text: PAT trees and PAT arrays

Gaston H. Gonnet
Dept. of Computer Science
ETH, Zurich
Switzerland

Ricardo A. Baeza-Yates
Depto. de Ciencias de la Computación
Universidad de Chile
Casilla 2777,
Santiago, Chile

Tim Snider
Centre for the New OED and Text Research
University of Waterloo
Waterloo, Ontario
Canada N2L 3G1

Abstract

We survey new indices for text, with emphasis on PAT arrays (also called suffix arrays). A PAT array is an index based on a new model of text which does not use the concept of word and does not need to know the structure of the text.

1 Introduction

Text searching methods may be classified as lexicographical indices (indices that are sorted), clustering techniques, and indices based on hashing. In this chapter we discuss two new lexicographical indices for text, called PAT trees and PAT arrays. Our aim is to build an index for the text of size similar to or smaller than the text.

Briefly, the traditional model of text used in information retrieval is that of a *set of documents*. Each document is assigned a list of *keywords* (attributes), with optional relevance *weights* associated

to each keyword. This model is oriented to library applications, which it serves quite well. For more general applications it has some problems, namely:

- A basic structure is assumed (documents and words). This may be reasonable for many applications, but not for others.
- Keywords must be extracted from the text (this is called "indexing"). This task is not trivial and error prone, whether it is done by a person, or automatically by a computer.
- Queries are restricted to keywords.

For some indices, instead of indexing a set of keywords, all words except for those deemed to be too common (called *stopwords*) are indexed.

We prefer a different model. We see the text as one long *string*. Each position in the text corresponds to a semi-infinite string (*sistring*), the string that starts at that position and extends arbitrarily far to the right, or to the end of the text. It is not difficult to see that any two strings not at the same position are different. The main advantages of this model are:

- No structure of the text is needed, although if there is one, it can be used.
- No keywords are used. The queries are based on *prefixes* of *sistrings*, that is, on any substring of the text.

This model is simpler and does not restrict the query domain. Furthermore, almost any searching structure can be used to support this view of text.

In the traditional text model each document is considered a database record, and each keyword a value or a secondary-key. Because the number of keywords is variable, common database techniques are not useful in this context. Typical database queries are on equality or on ranges. They seldom consider "approximate text searching".

This paper describes PAT trees and PAT arrays. PAT arrays are an efficient implementation of PAT trees, and support a query language more powerful than do traditional structures based on keywords and boolean operations. PAT arrays were independently discovered by Gonnet (1987) and Manber and Myers (1990). Gonnet used them for the implementation of a fast text searching system, PATTM (see Gonnet (1987) and Fawcett (1989)), used with the *Oxford English Dictionary*

(*OED*). Manber and Myers motivation was searching in large genetic databases. We will explain how to build and how to search PAT arrays.

2 The PAT tree structure

The PAT tree is a data structure that allows very efficient searching with preprocessing. This section describes the PAT data structure, how to do some text searches and algorithms to build two of its possible implementations. This structure was originally described by Gonnet in the paper "Unstructured Data Bases" by Gonnet (1983). In 1985 it was implemented and later used in conjunction with the computerization of the *Oxford English Dictionary (OED)*. The name of the implementation, the PATTM system, has become well known in its own right, as a software package for very fast string searching.

2.1 Semi-infinite strings

In what follows, we will use a very simple model of text. Our text, or text database will consist of a single (possibly very long) array of characters, numbered sequentially from one onwards. Whether the text is already presented as such, or whether it can be viewed as such is not relevant. To apply our algorithms it is sufficient to be able to view the entire text as an array of characters.

A semi-infinite string is a subsequence of characters from this array, taken from a given starting point but going on as necessary to the right. In case the semi-infinite string (sistring) is used beyond the end of the actual text, special null characters will be considered to be added at its end, these characters being different than any other in the text. The name semi-infinite is taken from the analogy with geometry where we have semi-infinite lines, lines with one origin, but infinite in one direction. Sistrings are uniquely identified by the position where they start, and for a given, fixed text, this is simply given by an integer.

Example:

Text Once upon a time, in a far away land ...

sistring 1 Once upon a time ...

sistring 2	nce upon a time ...
sistring 8	on a time, in a ...
sistring 11	a time, in a far ...
sistring 22	a far away land ...

Sistrings can be defined formally as an abstract data type and as such present a very useful and important model of text. For the purpose of this section, the most important operation on sistrings is the lexicographical comparison of sistrings and will be the only one defined. This comparison is the one resulting from comparing two sistrings' contents (not their positions). Note that unless we are comparing a sistring to itself, the comparison of two sistrings cannot yield equal. (If the sistrings are not the same, sooner or later, by inspecting enough characters, we will have to find a character where they differ, even if we have to start comparing the fictitious null characters at the end of the text).

For example the above sistrings will compare as follows:

$$22 < 11 < 2 < 8 < 1$$

Of the first 22 sistrings (using ASCII ordering) the lowest sistring is "a far away ..." and the highest is "upon a time ...".

2.2 PAT tree

A PAT tree is a Patricia tree (see Morrison (1968), Knuth (1973), Flajolet and Sedgewick (1986), and Gonnet (1988)) constructed over all the possible sistrings of a text. A Patricia tree is a digital tree where the individual bits of the keys are used to decide on the branching. A zero bit will cause a branch to the left subtree, a one bit will cause a branch to the right subtree. Hence Patricia trees are binary digital trees. In addition, Patricia trees have in each internal node an indication of which bit of the query is to be used for branching. This may be given by an absolute bit position, or by a count of the number of bits to *skip*. This allows internal nodes with single descendants to be eliminated and thus all internal nodes of the tree produce a useful branching, i.e. both subtrees are non-null. Patricia trees are very similar to compact suffix trees or compact position trees (see Aho *et al.* (1974)).

Patricia trees store key values at external nodes, the internal nodes have no key information, just the skip counter and the pointers to the subtrees. The external nodes in a PAT tree are sistrings, i.e. integer displacements. For a text of size n , there are n external nodes in the PAT tree and $n - 1$ internal nodes. This makes the tree $O(n)$ in size, with a relatively small asymptotic constant. Later we will want to store some additional information (the size of the subtree and which is the taller subtree) with each internal node but this information will always be of a constant size.

Figure 1 shows an example of a PAT tree over a sequence of bits (normally it would be over a sequence of characters), just for the purpose of making the example easier to understand. In this example, we show the Patricia tree for the text "01100100010111..." after the first 8 sistrings have been inserted. External nodes are indicated by squares, and they contain a reference to a sistring, internal nodes are indicated by a circle and contain a displacement. In this case we have used, in each internal node, the total displacement of the bit to be inspected, rather than the skip value.

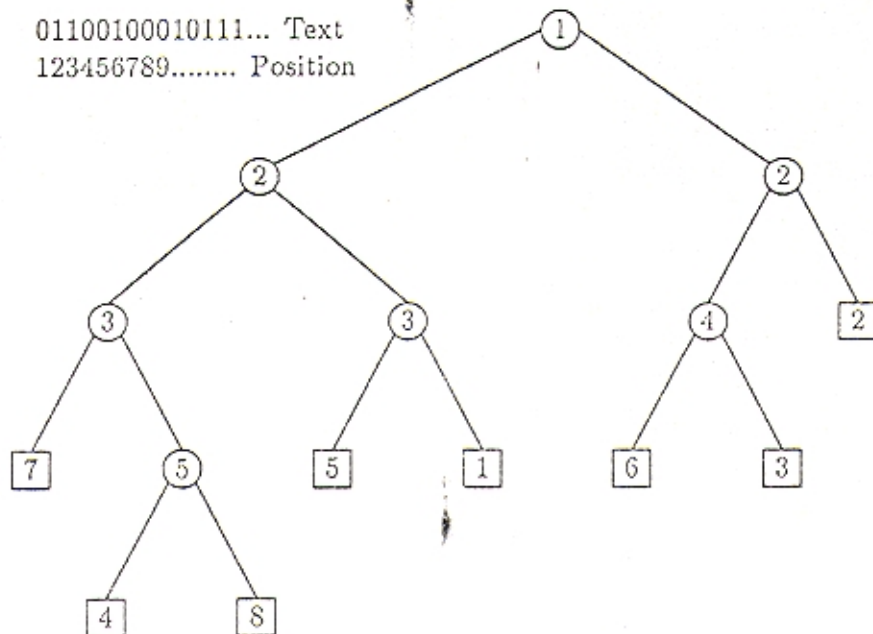


Figure 1: Pat tree when the sistrings 1 through 8 have been inserted.

Notice that to reach the external node for the query 00101 we first inspect bit 1, (it is a zero we go left) then bit 2 (it is zero, we go left), then bit 3 (it is a one, we go right) and then bit 5 (it is a one, we go right). Because we may skip the inspection of some bits (in this case bit 4), once we reach our desired node we have to make one final comparison with one of the sistrings stored