Implementation of the Substring Test by Hashing

Malcolm C. Harrison
New York University*

A technique is described for implementing the test which determines if one string is a substring of another. When there is low probability that the test will be satisfied, it is shown how the operation can be speeded up considerably if it is preceded by a test on appropriately chosen hash codes of the strings.

Key Words and Phrases: substring, hashing, subset, signature, information compression, information retrieval, searching

CR Categories: 3.74, 5.30, 5.6

* Courant Institute of Mathematical Sciences, 251 Mercer Street, New York, NY 10012. This work was done under AEC contract number AT (30-1) 1480 V.

This note describes a fast implementation of the test which determines if one string contains a specified substring. The scheme makes use of hashing techniques and of the ability to do many Boolean operations in parallel on a standard computer. It is most useful when the same strings are being tested repeatedly, and when the probability of finding the substring is small.

The scheme makes use of three ideas. The first is that if the search is likely to be unsuccessful, it can usefully be preceded by a computationally faster test for necessary but not sufficient conditions that the substring be found. A simple example of such a test is the comparison of the lengths of the strings, but this is usually too weak a test to be useful.

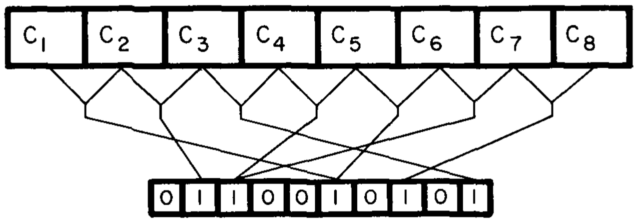
The second is that a string can be represented by the set of its substrings, and in particular by the set of its substrings of a specified length. In general such a representation is not unique, but it does preserve the substring property in the sense that, if one string has another string as a substring, the set of substrings of the first will include the set of substrings of the second. Because of the lack of uniqueness, the reverse is not true, of course.

The third is that a set S can be represented by a binary string $b_1b_2b_3 \cdots b_m$ in which a value of 1 for b_i indicates that S contains at least one element of the set E_i. In general such a representation is not unique, unless each E_i contains exactly one element and each possible element is contained in some E_i. However, it preserves the subset property in the sense that, if set S_1 is a subset of set S_2, the binary string representing S_2 will have ones in all positions where the string representing S_1 has ones.

Accordingly, consider a string S to be represented by a binary string $b_1b_2b_3 \cdots b_m$ constructed as follows:

1. Set all bits b_i to zero.

Fig. 1. 10-bit hashed 2-signature of the string $c_1c_2 \cdots c_8$



2. For each substring S of length k, compute i = hash(s) and set $b_i = 1$.

This uses a hashing function "hash" which is assumed to give an integer result in the range 1 to m. The resulting binary string, which contains a 1-bit only in positions which correspond to certain substrings of length k, is called the *hashed k-signature* of the string S. A hashed 2-signature is illustrated in Figure 1. It is clear that for any particular hashing function a necessary condition that string S_1 be a substring of string S_2 is that the hashed k-signature of S_2 have ones wherever the hashed k-signature of S_1 has ones. In practice it is often convenient to choose m so that the signature fits in a single machine word, in which case the signature test can be implemented in one or two machine instructions, or in FORTRAN as

```
IF ((KSIG1.A..N.KSIG2).NE.O) GOTO 100
IF (.NOT.SUBSTR(S1.S2)) GOTO 100
```

where SUBSTR is the rigorous test, and where we want to go to statement 100 if the substring is not found. The operators .A. and .N. are used to represent bit-by-bit "and" and "not" operations on binary words.

Choice of Parameters

Two parameters, m and k, are available to be chosen. As mentioned above, it is usually convenient to choose m so that it is a multiple of the number of bits in a machine word, but this is not necessary. Clearly the larger m is, the more accurate the results will be, in the sense that fewer strings will be incorrectly identified as substrings by the signature test. The more time required for the rigorous test, the more worthwhile an improvement in the signature test becomes. The amount of accuracy which is worth achieving also depends on the a priori probability that the string is not a substring—it is probably not worth implementing a signature which makes only 2 percent errors instead of 5 percent if 30 percent of the strings are in fact substrings. However, if only 1 percent of the strings are substrings such an improvement could be well worthwhile.

The parameter k can also be chosen. Clearly if k is 1, no information is included in the signature about the order, so we will ordinarily expect k to be two or more. If there are n possible symbols in the strings, it does not make sense to choose k so small that n^k is less than m, since there are only n^k distinct substrings of length k, and bits in the signature will be wasted. On the other hand, if k is chosen too large, the number of bits in the signature can become too small, and in fact will be zero if k is larger than the length of the string. Note that maximum information content corresponds to having about half the bits in the signature zero.

If reasonable choices are assumed for the parameters m and k, and random strings and a random hash function, the expectation value of the numbers of zeros in the signature of a string of length $l + k - 1$ is approximately $me^{-l/m}$ (see Feller [1, p. 58, ex. 6]).[1] Therefore, the a priori probability that a string of length $l + k - 1$ will be identified as a substring of a string of length $l_2 + k - 1$ will be

$$p(l_1, l_2, m) = (1 - e^{-l_2/m})^{m-me^{-l_1/m}}$$

[1] The distribution of ones in a hashed k-signature is not strictly Poisson, since not all sets of substrings correspond to strings, but the error in making this assumption is small.

Table I. A priori probability that a random string of length $l_1 + k - 1$ will be identified as a substring of a string of length $l_2 + k - 1$ using 32-bit hashed k-signatures.

l_2								
204.8	.99495	.99040	.98261	.97112	.95842	.95024	.94826	
102.4	.88097	.78553	.64465	.48030	.34547	.27875	.26461	
51.2	.50321	.27032	.09263	.01880	.00315	.00099	.00074	
25.6	.16254	.03141	.00185	.00003	.00000	.00000	.00000	
12.8	.03408	.00160	.00001	.00000	.00000	.00000	.00000	
6.4	.00551	.00005	.00000	.00000	.00000	.00000	.00000	
3.2	.00077	.00000	.00000	.00000	.00000	.00000	.00000	
	3.2	6.4	12.8	25.6	51.2	102.4	204.8	l_1

If d is the density of ones in the signature of the second string, that is $d = 1 - e^{-l_2/m}$, the a priori probability above is approximately d^{l_1} if l_1 is small compared with m, and approaches d^m as l_1 gets greater than m.

To illustrate the type of results which might be expected, this a priori probability is tabulated in Table I for a signature of 32 bits. This shows, for example, that the probability of a 13-character string being identified as a substring of a 52-character string is less than 10 percent, using 2-signatures. This implies that at least 90 percent of nonsubstrings will be rejected by the signature test. Also, since

$$p(l_1, l_2, m) = p^2(l_1/2, l_2/2, m),$$

doubling the length of the 2-signature to 64 bits will reduce the probability to about 0.1 percent, giving 99.9 percent rejection of nonsubstrings.

This technique has been used effectively in a conversational editor written by Richard Reich for the CDC 6600, which permits a line to be located by specifying a substring of characters. The editor then searches the file for such a subsequence. By storing with each line its hashed 2-signature, this search procedure can be speeded up considerably. There are clearly many other similar applications.

As we pointed out, the technique involves three separate notions, which can of course be used in many other ways. For example, any unordered set can be represented by a hashed signature constructed from its elements, and then the comparison of signatures can be used as a preliminary to the subset test. This could be useful in information retrieval situations where a document is described by a set of properties and where all those documents with a specified subset of properties are to be retrieved.

The essential property of these methods is that a complex data-object is represented by a simpler data-object which contains less information but which retains some of the properties of the original. The hashing function and the representation of an ordered set by its set of k-sequences should accordingly be considered as information compression functions which preserve as much of the relevant information on the data-object as possible. It seems likely that there is a more general theory of such functions, of which the example given above is only a special case.

Received March 1971

References

1. Feller, W. *An Introduction to Probability Theory and its Applications* (2nd ed.). Wiley, New York, 1957.
2. Harrison, M.C. Set comparison using hashing techniques. AEC Res. and Develop. Rep. NYO-1480-155, Courant Institute, New York University, New York, June 1970.