4/ 17

SEARCHING A DICTIONARY WITHIN TIGHTER TIME BOUNDS

D. S. Hirschberg

January 1979

Abstract:

An algorithm is given that determines whether a word is a member of a lexicographically ordered dictionary. This algorithm, an extension of both linear and binary search, has worst case time bounds that, depending upon the length of the words involved, may be less than half that of binary search.

Key Words: dictionary, searching, binary search, vector, complexity

CR Categories: 3.74, 5.25

Research supported by NSF grant MCS-76-07683. Author's current address: Department of Electrical Engineering, Rice University, Houston Texas 77001.

We consider the problem of searching a dictionary, each word consisting of at most k letters. It is assumed that the alphabet is ordered and that the dictionary is in lexicographic order.

Two letters can be compared (to determine which has lower rank in the alphabet) in unit time. Our goal is to exhibit an algorithm for searching the dictionary that has minimal worst case time complexity.

We denote the ordered set of words (k-vectors) by $\bar{B}$ and refer to the i-th word (vector) by B_i. Let $|\bar{B}| = n-1$. We wish to determine if the vector A is in $\bar{B}$.

Let B_0 be a vector lexicographically less than B_1 whose first x components were found to equal the corresponding components of A and, analogously, B_n has had its first y components found equal to the corresponding components in A. x and y are 0 initially.

Note that for all $1 \le i < n$ we need not compare $a_j : b_{ij}$ for $j \le \min(x,y)$ since the result must be "equal" from the results of previous comparisons and the fact that $\bar{B}$ is in lexicographic order.

Let T(x,y,n) be the minimum number of comparisons required, in the worst case, to search the n-1 vectors with x and y as defined above, using the following "quicksearch" algorithm.

```
function QUICKSEARCH(low,high,x,y)
n <-- high - low
if $n \le n_o$ (defined below) then use linear search [3]:
          j <-- low
          i <-- x+1
          while j<high AND $i \le k$
          do   compare $a_i : b_{ji}$
               if = then i <-- i + 1
                    else if > then j <-- j + 1
                         else return 0
          od
          if j=high then return 0
          if $a_i = b_{ji}$ for all $i \in \{y+1,\dots,k-1\}$
               then return j
               else return 0
else evaluate p (as described below) and:
          i <-- min{x,y} + 1
          while i<k AND $a_i = b_{pi}$ do i <-- i + 1
          compare $a_i : b_{pi}$
          if = then return p
          else if < then return QUICKSEARCH(low,p,x,i-1)
               else return QUICKSEARCH(p+1,high,i-1,y)
end.
```

If linear search is used, $T(x,y,n) = 2k + n-3 - x - y$.

Otherwise, $z = \min\{x,y\}$ and

if $A=B_p$ then $T(x,y,n) = k-z$

if $A<B_p$ then $T(x,y,n) = i-z + T(x,i-1,p)$

if $A>B_p$ then $T(x,y,n) = i-z + T(i-1,y,n-p)$.

$T(x,y,n)$ will be bound by the maximum of these cases.

Note that binary search is quicksearch with $p=n/2$ (and $n_o=1$) and that modified binary search (described in [4]) is quicksearch with $p=n/4$.

Assume that $T(x,y,n) = C(n) - x - y$. Thus,

$$C(n) \le \max\{C(p)+1+y-z,\ C(n-p)+1+x-z\}$$

To minimize the maximum (employing the technique of balancing [1]), we should endeavor to cause both possibilities to have equal complexity. Assuming $C(n) = (k/\log k)\log n$* [this assumption will be borne out], if $y \ge x$ we should choose $p = n/(1 + 2^{(y-x)(\log k)/k})$. However, since $T(x,y,n)$ with k implicit is the same as $T(0,y-x,n)$ with $k-x$ implicit, we suggest choosing $p = n/(1 + 2^{(y-x)\log(k-x)/(k-x)})$. A similar result will hold if $y<x$. In particular, if $y=x$ then $p=n/2$, i.e. binary search.

What is a good cutoff value n_0? When should we use linear search? The worst case of quicksearch is better than that of binary search. Using binary search and then linear search results in

$$C(n) \le C(n/2) + 1 + y-x \le C(n/2) + k$$
$$\le 3k - 3 + n/2$$

whereas using linear search directly results in

$$C(n) \le 2k + n - 3.$$

Thus, linear search should not be used if $n>2k$. A more detailed analysis would indicate a tighter cutoff. Empirical results indicate that $n_0=1.5k$ is a good choice.

---

*Unless specified otherwise, logarithms are base 2.

Using values of k=6 and 20, we have obtained results indicating savings of over 25 and 50 percent respectively on the number of comparisons required in the worst case as compared to binary search.

We also ran simulation experiments to compare these algorithms for "average case" behavior. Assuming all strings in the dictionary have equal probability of being queried, the following percentage savings were observed for average observed case and worst observed case number of comparisons (obtained by using quicksearch as opposed to binary search). For a dictionary containing the sequence of all binary strings of length 14, 53.1 and 26.5; for the sequence of all quatenary strings of length 7, 46.1 and 27.4; for the sequence of every 41st string in the set of quatenary strings of length 10, 45.9 and 26.9.

In running these experiments, rather than compute complicated logarithms and exponentials, we used the approximation p = low + n/(1 + DELTA), where DELTA = max(1.4,y-x) when y>x and DELTA = 1 when y=x. When y<x, p = high - n/(1 + DELTA) using an analagous computation of DELTA.

The complexity of this algorithm is $O(k \log_k n)$, as can be verified by substitution in the recurrence equations, and compares favorably to the k log n comparisons required by binary search [2]. Detailed analysis to obtain the exact coefficients is left as an open problem.

References

1. Aho A.V., Hopcroft J.E., and Ullman J.D. The Design and Analysis of Computer Algorithms. Addison-Wesley, 1974.

2. Dobkin D. and Lipton R.J. Multidimensional searching problems. SIAM J.Computing 5:2 (June 1976), pp.181-186.

3. Hirschberg D.S. On the complexity of searching a set of vectors. SIAM Jour. on Computing (to appear).

4. Hirschberg D.S. A lower worst-case complexity for searching a dictionary. Proc. 16th Allerton Conference, October 1978.

5. Knuth D.E. The Art of Computer Programming, Vol. 3. Addison-Wesley, 1973, pp.200-204.

6. Raghavan V.V. and Yu C.T. A note on a multidimensional searching problem. IPL 6:4 (Aug. 1977), pp.133-135.