A LOWER WORST-CASE COMPLEXITY FOR SEARCHING A DICTIONARY

D.S. HIRSCHBERG
Rice University
Electrical Engineering Department
Houston, Texas

ABSTRACT

It is shown that $k(p+3)/2 + p-2$ letter comparisons suffice to determine whether a word is a member of a lexicographically ordered dictionary containing $2^p - 1$ words of length k. This offers a potential savings (compared to worst case complexity of binary search) that asymptotically approaches 50 percent.

We consider the problem of searching a dictionary containing n words, 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 $\mathbb{B}$ and refer to the i-th word (vector) by B_i. We wish to determine if the vector A is in $\mathbb{B}$. Assume that $n = 2^p - 1$, i.e. $p = \lceil \log(n+1) \rceil$.

Let B_m be the "middle" vector ($m = 2^{p-1}$), let B_q be the "quarter" vector ($q = 2^{p-1} + 2^{p-2}$).

Using binary search, i.e. comparing A first with B_m and then, depending upon the result, comparing A with either B_q or with B_r etc. will lead to a worst case complexity of kp comparisons.

In a recent paper [2], Hirschberg has exhibited an algorithm, VECTOR-SEARCH, that has a worst case complexity of less than $k(p+2-\log k)$. For many values of n and k, this algorithm offers a potential savings of from 5 to 25 percent.

Research supported by NSF grant MCS-76-3933. Author's current address: Department of Electrical Engineering, Rice University, Houston, Texas, 77001.

Let there have been x "equal" comparisons between A and B_0 (if no comparisons were made between A and B_0 then x = 0) and y "equal" comparisons between A and B_{n+1}. Without loss of generality, assume that $x \leq y$ (the case $x > y$ will lead to symmetric arguments).

For the algorithm we shall next consider, which we call <u>modified binary search</u>, let T(x,y,p) be the minimum number of comparisons required for x, y, and p as defined above and for k (the length of the vectors) implicit. We shall demonstrate a savings that asymptotically approaches 50 percent of the worst case complexity as compared to the binary search method.

Note that for all $1 \leq i \leq n$ we need not compare $a_j : b_{ij}$ for $j \leq x$ since the result <u>must</u> be "equal" from the results of previous comparisons and the fact that $\overline{B}$ is in lexicographic order.

Our algorithm first compares components of A with components of B_q (starting at component x + 1) until it is determined whether $A < B_q$, $A = B_q$, or $A > B_q$.

<u>Case 1</u>. If $A < B_q$ because $a_{t+1} < b_{q,t+1}$ $(x \leq t < k)$ then

$$T(x,y,p) = T(x,t,p-2) + t+1-x \tag{1}$$

<u>Case 2</u>. If $A > B_q$ because $a_{t+1} > b_{q,t+1}$ then compare A with another vector depending on the value of t.

<u>Case 2a</u>. If $t \leq y$ then compare A with B_m (starting with component t + 1)

<u>2aa</u>. If $A < B_m$ because $a_{h+1} < b_{m,h+1}$ $(x \leq t \leq h < k)$ then

$$T(x,y,p) = T(t,h,p-2) + h+2-x \tag{2}$$

<u>2ab</u>. If $A > B_m$ because $a_{h+1} > b_{m,h+1}$ $(x \leq t \leq h < k)$ then

$$T(x,y,p) = T(\min\{h,y\}, \max\{h,y\}, p-1) + h+2-x \tag{3}$$

<u>2ac</u>. If $A = B_m$ then

$$T(x,y,p) = k+1-x \tag{4}$$

<u>Case 2b</u>. If $t \geq y$ then compare A with B_r (starting with component y+1)

2ba. If $A < B_r$ because $a_{h+1} < b_{r,h+1}$ $(y \leq h,\ t < k)$ then

$$T(x,y,p) = T(\min\{h,t\},\ \max\{h,t\},\ p-1) + t+1-x + h+1-y \qquad (5)$$

2bb. If $A > B_r$ because $a_{h+1} > b_{r,h+1}$ $(y \leq h,\ t < k)$ then

$$T(x,y,p) = T(y,h,p-2) + t+1-x + h+1-y \qquad (6)$$

2bc. If $A = B_r$ then

$$T(x,y,p) = t+1-x + k-y \qquad (7)$$

Case 3. If $A = B_q$ then

$$T(x,y,p) = k-x \qquad (8)$$

The resulting problem has either half or quarter the range (in $\overline{B}$) of the original problem. If the range is reduced to zero then terminate with the solution that A is not in $\overline{B}$. If the range consists of one vector B then compare $a_{x+1} \ldots a_k$ with the corresponding k-x components of B. Otherwise, recursively apply the modified binary search algorithm.

Theorem. For $p \geq 1$, $T(x,y,p) \leq k(p+3)/2 + p-2 - x - y$

Proof. By induction on p. Note that $T(x,y,0) = 0$. The theorem, for $p = 1$, is true since $T(x,y,1) = k-x$ (the algorithm treats this case separately). The theorem, for $p = 2$, is also true as can be seen by substituting for $T(\cdot,\cdot,0)$ and $T(\cdot,\cdot,1)$ in the righthand side of the formulas given in all of the cases, one of which must occur.

For the inductive step, T(x,y,p) is bounded from above by the maximum of the cases. In all cases, substituting in the assumed complexity for T (with smaller value of the third argument) substantiates that the theorem holds.

The table below illustrates typical percentage savings (worst case analysis) of VECTOR-SEARCH and modified binary search as compared to binary search.

n (p)	k	Binary search	VECTOR-SEARCH (%)	modified bin. search (%)
125 (7)	4	28	26 (7.1)	25 (10.7)
	8	56	46 (17.9)	45 (19.6)
	16	112	78 (30.4)	81 (27.7)
1000 (10)	4	40	38 (5.0)	34 (15.0)
	8	80	70 (12.5)	60 (25.0)
	16	160	126 (21.2)	112 (30.0)
10^6 (20)	4	80	78 (2.5)	64 (20.0)
	8	160	150 (6.2)	110 (31.2)
	16	320	286 (10.6)	202 (36.9)

We have shown that, approximately, $k(\log_2 n)/2$ letter comparisons suffice to search a dictionary. Research currently in progress indicates that this upper bound on the complexity of dictionary searching may be reduced to $k(\log_2 n)/\log_2 k$. A lower bound of $k + \log_2 n$ can easily be shown (see [2]) and it is an open problem to narrow the gap between these bounds.

REFERENCES

1. D. Dobkin and R.J. Lipton. Multidimensional Searching Problems. SIAM J. Computing 5:2 (June 1976), pp. 181-186.
2. D.S. Hirschberg. On the Complexity of Vector Searching. Rice University Department of Electrical Engineering Technical Report #7807, June 1978.
3. D.E. Knuth. The Art of Computer Programming, Vol. 3. Addison-Wesley, 1973, pp. 200-204.
4. V.V. Raghavan and C.T. Yu. A Note on a Multidimensional Searching Problem. IPL 6:4 (August 1977), pp. 133-135.