SIAM J. COMPUT.
Vol. 6, No. 4, December 1977

ON THE WORST-CASE BEHAVIOR OF STRING-SEARCHING ALGORITHMS*

RONALD L. RIVEST†

Abstract. Any algorithm for finding a pattern of length k in a string of length n must examine at least $n-k+1$ of the characters of the string in the worst case. By considering the pattern $00\cdots 0$, we prove that this is the best possible result. Therefore there do not exist pattern matching algorithms whose worst-case behavior is "sublinear" in n (that is, linear with constant less than one), in contrast with the situation for average behavior (the Boyer–Moore algorithm is known to be sublinear on the average).

Key words. string-searching, pattern matching, text editing, computational complexity, worst-case behavior

1. Introduction. Let $s=s_1s_2\cdots s_n$ denote a string of length n over some finite alphabet Σ, and similarly let $p=p_1p_2\cdots p_k$ denote a pattern of length k over the same alphabet. The "string-searching problem" is to determine if the pattern occurs in the string—that is, if

$$(\exists\, j)(1\leqq j\leqq n-k+1)\wedge(p_1p_2\cdots p_k=s_js_{j+1}\cdots s_{j+k-1}).$$

We denote this occurrence as $p\leqq s$.

Several efficient algorithms exist for determining whether $p\leqq s$, given a pattern p of length k and a string s of length n. For example, the algorithm of Knuth, Morris and Pratt [3], [4] first constructs (in time $O(k)$) a finite state automaton to recognize the regular set $\Sigma^*p\Sigma^*$ (see [1] also). Then $p\leqq s$ iff the automaton accepts s, which can be determined in time $O(n)$. The entire algorithm runs in time $O(n+k)$. As an example (which we shall use later), for $p=0101$ the automaton of Fig. 1 would be constructed. Here we assume that $\Sigma=\{0, 1\}$. State 1 is the initial state and state 5 is the only accepting state.

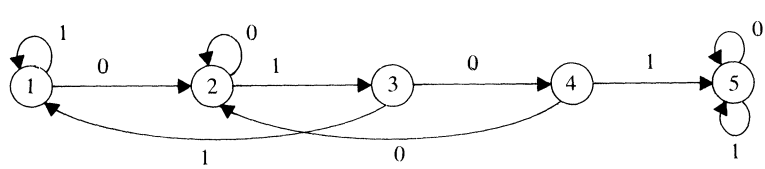

FIG. 1

Recently, Boyer and Moore published an algorithm [2] which is significantly faster than the Knuth–Morris–Pratt algorithm on the average. The latter algorithm examines every character in s exactly once, whereas the Boyer–Moore algorithm looks at only some fraction $c<1$ of the characters on the average; a

* Received by the editors June 15, 1976, and in revised form October 5, 1976.

† Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139. This report was prepared with the support of the National Science Foundation Grant GJ-43634X, Contract DCR74-12997-A01, and Grant MCS76-14294.

typical value for c might be .24 when p is a five-letter English word. The worst-case behavior of the algorithm is nonlinear in n and k, although a slight modification of their algorithm due to B. Kuipers results in a linear worst-case time algorithm as well. (Knuth [5] has shown that the average number of times a character in s is examined by the modified algorithm is bounded above by 6; the proof, however, is *very* complicated.) The Boyer–Moore algorithm requires that the string s be stored in some sort of random-access memory in order to achieve any savings. Their procedure examines s_k, then s_{k-1}, and so on, until an s_j such that $s_j \neq p_j$ is found. Then some of the initial characters of s may be deleted and the process repeated with the shorter string s. If the examined (matching) subsection $s_{j+1} \cdots s_k$ of s occurs nowhere else in p, the first k characters of s may be skipped, even though only $k-j+1$ of them have been examined. Otherwise some smaller number may be discarded, reflecting the next possible alignment of $s_{j+1} \cdots s_k$ with some subsection of p. Another heuristic is also used: the latest occurrence of s_j in p (hopefully preceding p_j) is used to determine how many characters from s can be deleted before s_j aligns with some character in p. In the best case we find that $s_k \neq p_k$ and that s_k occurs nowhere in p; then k characters of s can be skipped at the cost of examining just one.

The focus of this paper is on the worst-case behavior of such pattern-matching algorithms. We answer (in the negative) the conjecture that a pattern-matching algorithm can exist whose worst-case behavior is "sublinear" in the same sense that the Boyer–Moore algorithm is sublinear in its average behavior. More precisely, we show that for every pattern p and for every correct algorithm A which determines if $p \leqq s$ for arbitrary strings s, there exists a string s which causes A to examine at least $|s|-|p|+1$ characters of s. This result is given in § 2 of this paper. In § 3 we show that this lower bound is the best possible by considering an algorithm for the pattern $p = 00 \cdots 0$.

2. The worst-case lower bound. The approach models the method Rivest and Vuillemin used to prove the Aanderaa–Rosenberg conjecture [5]. Fix the pattern p and let A_p be any algorithm for determining whether $p \leqq s$ for any string s. Let $w(A_p, n)$ denote the maximum number of characters in s examined by algorithm A_p for any string s in Σ^n; $w(A_p, n)$ is the worst-case cost function for algorithm A.

We assume that $w(A_p, n) \leqq w(A_p, n+1)$ for all A_p and all n. Otherwise if $w(A_p, n) > w(A_p, n+1)$ for some n an improved algorithm A'_p can be derived from A_p by letting A'_p behave on inputs s just as A_p does whenever $|s| \neq n$ and letting A'_p behave on the strings s of length n just as A_p would behave on the string sz where $z \neq p_k$ (simulating the query of the $(n+1)$st character z). Since $(p \leqq s) \Leftrightarrow (p \leqq sz)$, we have that $w(A'_p, n) \leqq w(A_p, n+1)$, and $w(A'_p, m) = w(A_p, m)$ for $m \neq n$. Thus $w(A'_p, n) \leqq w(A'_p, n+1)$; repeating this procedure as necessary yields an improved procedure A'_p such that $w(A'_p, n) \leqq w(A_p, n)$ for all n and $w(A'_p, n) \leqq w(A'_p, n+1)$ for all n.

THEOREM 1. $(\forall p)(\forall A_p)(\forall n)(w(A, n) \geqq n-k+1)$, *where* $k = |p|$.

Proof. We shall in fact prove that $w(A_p, n) = n$ for infinitely many n, such that these values of n occur not more than k apart. Using our assumption that $w(A_p, n) \leqq w(A_p, n+1)$ then yields the theorem.

Let $f(p, n)$ denote $|\{s \mid s \in \Sigma^n \wedge p \leqq s\}|$, the number of strings of length n which contain p as a substring. The following result is immediate from [5].

LEMMA 1. *If* $f(p, n) \not\equiv 0 (\mathrm{mod} |\Sigma|)$, *then* $w(A_p, n) = n$.

The proof of Lemma 1 will not be given here; we only remark that it follows from a calculation of $f(p, n)$ using a decision-tree representation of A_p. If $w(A_p, n) < n$ then $f(p, n) \equiv 0 \pmod{|\Sigma|}$ follows.

In order to calculate $f(p, n)$ we make use of the finite state automaton (fsa) constructed by the Knuth–Morris–Pratt algorithm for recognizing $\Sigma^* p \Sigma^*$. Let the states of this fsa be numbered so that state 1 is the initial state, state i (for $1 \leqq i \leqq k$) is arrived at whenever a string ending in $p_1 p_2 \cdots p_{i-1}$ has been read (and this is the largest such i), and state $k+1$ is the accepting state. There is a transition labeled p_i from state $i-1$ to state i (for $1 \leqq i \leqq k$); all other transitions leaving state $i-1$ arrive at some state numbered strictly less than i.

Let $g_p(n, i)$ denote $|\{s \mid s \in \Sigma^n$ and the fsa on s ends in state $i\}|$. Then $g_p(n, k+1) = f(p, n)$. The fsa will be used to derive a set of linear recurrences for the vector $\bar{g}_n = (g_p(n, 1), g_p(n, 2), \cdots, g_p(n, k+1))$. In fact $\bar{g}_{n+1} = T \cdot \bar{g}_n$, where T is a $k+1$ by $k+1$ matrix whose (i, j) entry is the number of symbols in Σ which cause a transition from state j to state i. For example, for $p = 0101$ the corresponding matrix $T = \{t_{ij}\}$ is

$$\begin{bmatrix} 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 2 \end{bmatrix}.$$

In general, the sum of each column is $|\Sigma|$, $t_{i,i-1} = 1$ for $2 \leqq i \leqq k+1$, and $t_{ij} = 0$ if $j < i-1$. Also, $t_{k+1,k+1} = |\Sigma|$. To initialize the recurrence we have $\bar{g}_0 = (1, 0, 0, \cdots, 0)$.

Since we are interested in $f(p, n) = g_p(n, k+1)$ only with respect to its residue modulo $|\Sigma|$, we consider the reduced recurrence $\bar{g}_{n+1} = T' \cdot \bar{g}_n \pmod{|\Sigma|}$, where the entries of T' are those of T reduced modulo $|\Sigma|$. In fact T' is just T with the $(k+1, k+1)$ entry replaced by 0. We now observe that $g_p(n, k+1) \equiv g_p(n-1, k)$, so we will concentrate on the parity of $g_p(n, k)$ from now on. The k by k upper left submatrix T'' of T' maps $(g_p(n, 1) \pmod{|\Sigma|}, \cdots, g_p(n, k) \pmod{|\Sigma|})$ onto $\bar{g}'_{n+1}$.

Now T'' induces a mapping from $\Gamma = \{0, 1, \cdots, |\Sigma|-1\}^k$ to itself. Furthermore, T'' is easily seen to be invertible; sequentially adding row i to row $i+1$ for $i = 1, 2, \cdots, k$ will reduce T'' to an upper triangular form with $|\Sigma|-1$ along the main diagonal (we assume that $|\Sigma| > 1$).

Since T'' is invertible, the directed graph G, whose vertices are elements of Γ and whose edges (x, y) are present whenever $T''x = y$, consists of a set of disjoint cycles. We need to show that the cycle containing $\bar{g}'_0 = (1, 0, 0, \cdots, 0)$ has a vertex whose kth coordinate is nonzero at least once every k steps.

We first observe that the all-zero vector 0^k is not an element of the cycle, since it belongs to a one-element cycle (it is fixed by the linear mapping T''), and $\bar{g}'_0$ is not the zero vector.

Next we observe that for any vector $x \in \Gamma$ such that $x_i \neq 0$ and $x_j = 0$ for $j > i$, the vector $y = (T'')^{k-i}x$ has $y_k \neq 0$. In general, if $x \neq 0^k$, $x_k = 0$ and i is the largest integer such that $x_i \neq 0$, then $(T''x)_{i+1} = x_i$ since the lower diagonal portion of T'' is zero except for the subdiagonal, which consists entirely of ones.

This completes the proof of

LEMMA 2. $(\forall n > k)(\exists j)(0 \leqq j < k)(w(A_p, n-j) = n-j)$.

Theorem 1 now follows directly from Lemma 1 and our assumption. □

3. An upper bound on the worst-case. The lower bound of $|s| - |p| + 1$ proved in the last section may seem weak at first; one's first guess might be that $w(A_p, n) = n$ as long as $n \geqq |p|$. This, however, turns out to be false, as we demonstrate in this section by a careful analysis of an algorithm for the pattern $p = 0^k$.

THEOREM 2. $(p = 0^k) \Rightarrow (\exists A_p)(w(A_p, n) = n - \mu(n))$, *where*

$$\mu(n) = \begin{cases} 0 & \textit{if } n^p \equiv 0 \pmod{k+1} \textit{ or } n \equiv k \pmod{k+1}, \\ n \pmod{k+1} & \textit{otherwise.} \end{cases}$$

Proof. The algorithm A_p works in a fashion similar to the Boyer–Moore algorithm. It is given below.

ALGORITHM A_p *for* $p = 0^k$.
Input: a string $s_1 s_2 \cdots s_n$.
Local variables: r, i, j
Procedure:

```
r: = 0; i: = 0; j: = 0;
repeat if r + k > n then
        begin print ("p ≰ s"); exit end;
        if s_{r+k-j} = 0 then j: = j + 1
        else begin r := r + k - j;
                   i := j;
                   j := 0
    end;
until i + j = k;
print ("p ≦ s at position", r + 1).
```

Inductively the algorithm knows at the top of the repeat loop that positions $s_{r+1}, s_{r+2}, \cdots, s_{r+i}$ and positions $s_{r+k-j+1}, \cdots, s_{r+k}$ are all zero; it next tests position s_{r+k-j} and adjusts r, i, and j accordingly. Let $c(m, i, j)$ denote the maximum number of characters in s that A_p needs to examine, starting from some instant when $m = n - r$ and i and j define the state of A_p's knowledge about s as above. Thus $w(A_p, n) = c(n, 0, 0)$ by definition. Furthermore, we have by construction that

$$(*) \qquad c(n, i, j) = \begin{cases} 0 & \text{if } i + j = k \text{ or } n < k, \\ \max(c(n, i, j+1), c(n-k+j, j, 0)) + 1 & \text{otherwise.} \end{cases}$$

Define for integers m and i, $0 \leqq i \leqq k-1$, $0 \leqq m \leqq k$,

$$\beta(m, i) = \begin{cases} 0 & \text{if } m = k, \\ m+1 & \text{if } i > m \text{ and } m < k, \\ m - i & \text{if } i \leqq m \text{ and } m < k. \end{cases}$$

LEMMA.

$$c(n,i,j)=\begin{cases}0 & \text{if } i+j=k \text{ or } n<k,\\ n-i-j-\beta(m,i) & \text{otherwise, where } m=n \ (\mathrm{mod}\ k+1).\end{cases}$$

Proof. By induction, as in the definition (*) of $c(n, i, j)$. The lemma is clearly correct if $i+j=k$ or $n<k$. Henceforth, assume $i+j<k\leqq n$. There are two cases to consider. Let m denote n (mod $k+1$).

Case 1. $c(n,i,j)=c(n,i,j+1)+1$. Here the lemma follows directly as long as $i+j+1\neq k$; otherwise $c(n,i,j+1)\leqq c(n-k+j,j,0)$, so here we can appeal to Case 2.

Case 2. $c(n,i,j)=c(n-k+j,j,0)+1$.

Case 2a. $n-k+j<k$. Here we know that $c(n,i,j+1)\geqq c(n-k+j,j,0)$ so the lemma holds by Case 1. (If both $n-k+j<k$ and $i+j+1=k$ then the lemma follows by the definition of β).

Case 2b. $n-k+j\geqq k$.

Case 2b(1). $i+j+1=k$. Here we need to show that

$$n-i-j-\beta(m,i)=n-k+1-\beta(n-k+j\ (\mathrm{mod}\ k+1),j),$$

or

$$(**)\qquad \beta(m,i)=\beta(n-i-1\ (\mathrm{mod}\ k+1),k-1-i).$$

Case 2b(1)i. $m=k$. Here both sides of (**) are 0, since $n-i-1\equiv k-i-1\ (\mathrm{mod}\ k+1)$.

Case 2b(1)ii. $i>m$ and $m<k$. Both sides of (**) are $m+1$, since $n-i-1\ (\mathrm{mod}\ k+1)>k-i-1$.

Case 2b(1)iii. $i\leqq m$ and $m<k$. Both sides of (**) are $m-i$, since $0\leqq m-i-1<k-i-1$.

Case 2b(2). $i+j+1<k$. Here it suffices to show that

$$n-i-j-1-\beta(m,i)\geqq n-k+j-h-\beta(n-k+j\ (\mathrm{mod}\ k+1),j),$$

that is, that $c(n,i,j+1)\geqq c(n-k+j,j,0)$, so that we may appeal to Case 1. Since $m+1\equiv n-k\ (\mathrm{mod}\ k+1)$, this is equivalent to

$$(***)\qquad k-i-j\geqq 1+\beta(m,i)-\beta(m+j+1\ (\mathrm{mod}\ k+1),j).$$

Note that the left-hand side of (***) is strictly greater than one, since we are in Case 2b(2).

Case 2b(2)i. $m=k$. Here the right side of (***) is at most one.

Case 2b(2)ii. $i>m$. The right side of (***) equals 1 since $m+j+1<i+j+1<k$.

Case 2b(2)iii. $i\leq m$. If $m+j+1<k$, then the right hand side of (***) is $-i$. If $m+j+1=k$ then it is $1+m-i=k-i-j$. If $m+j+1>k$ then if $m<k$ it is $1+(m-i)-(m+j+1-k-1+1)=k-i-j$; otherwise it is one.

This completes the proof of the lemma. Theorem 2 follows since $\beta(n\ (\mathrm{mod}\ k+1),0)=\mu(n)$. $\square$

We conclude from Theorem 2 that when searching for the pattern 0^k in a string $s\in\Sigma^n$, we only need to examine at most $n-k+1$ characters of s if $n\equiv k-1\ (\mathrm{mod}\ k+1)$. The uniform lower bound of theorem 1 can therefore not be improved. Note that the use of the pattern 0^k in Theorem 2 means that Theorem 1 is best possible even for a binary alphabet.

Conclusions. We have shown that pattern matching in strings is inherently linear (with constant 1) in the length of the string. An open problem is to prove the equivalent of Theorem 2 for all patterns:

$$(\forall p)(\exists A_p)(\overset{\infty}{\exists} n)(w(A_p, n) = n - |p| + 1).$$

Acknowledgment. I would like to thank Donna Brown for several helpful discussions, during one of which a weak version of Theorem 2 was observed. I would also like to thank Leo Guibas and Robert Floyd for communications regarding improved proofs of Theorems 1 and 2, respectively.

REFERENCES

[1] A. V. AHO, AND M. J. CORASICK, *Fast pattern matching; An aid to bibliographic search*, Comm. ACM, 18 (1975), pp. 333–340.

[2] R. S. BOYER AND J. STROTHER MOORE, *A fast string searching algorithm*, Tech. Rep. 3, Stanford Res. Inst., Mar. 1976. To appear, Comm. ACM.

[3] D. E. KNUTH, J. H. MORRIS AND V. R. PRATT, *Fast pattern matching in strings*, Computer Science Department Tech. Rep. CS-74-440, Stanford Univ., Stanford, CA, 1974.

[4] ———, *Fast pattern matching in strings*, this Journal, 6 (1977), pp. 323–350.

[5] R. L. RIVEST AND J. VUILLEMIN, *A generalization and proof of the Aanderaa-Rosenberg conjecture*, Proc. 7th SIGACT Symp. on Theory of Computing, pp. 6–11.