A New Linear-Time "On-Line" Algorithm for Finding the Smallest Initial Palindrome of a String

GLENN MANACHER

University of Illinois, Chicago, Illinois

ABSTRACT. Despite significant advances in linear-time scanning algorithms, particularly those based wholly or in part on either Cook's linear-time simulation of two-way deterministic pushdown automata or Weiner's algorithm, the problem of recognizing the initial leftmost nonvoid palindrome of a string in time proportional to the length N of the palindrome, examining no symbols other than those in the palindrome, has remained open. The present algorithm solves this problem, assuming that addition of two integers less than or equal to N may be performed in a single operation. Like the Knuth-Morris-Pratt algorithm, it runs in time independent of the size of the input alphabet. The algorithm as presented finds only even palindromes. However, an extension allows one to recognize the initial odd or even palindrome of length 2 or greater. Other easy extensions permit the recognition of strings $(ww^R)^*$ of even palindromes and of all the initial palindromes. It appears possible that further extension may be used to show that $(ww^R)^*$ is in a sense recognizable in real time on a reasonably defined random access machine.

KEY WORDS AND PHRASES: linear-time algorithm, on-line recognition, palindrome

CR CATEGORIES: 5.22, 5.25, 5.30

Introduction

A pioneering theorem of Cook's [2] states that there exists a linear-time simulation of a two-way deterministic pushdown automaton (DPDA) on a random access computer[1] capable of storing and retrieving the number n in one operation, where n is the length of the input string. Briefly, a two-way DPDA is like the well-known one-way DPDA that accepts on empty stack, except that at every stage the input head has the option of remaining stationary, advancing one square, or backing up one square. Examples of how such machines operate may illustrate their power. Daniel Chester discovered that such a machine can recognize $S = \{ww^Ru\}$, where w and u are strings on a finite alphabet, $w \neq \epsilon$, and R indicates reversal. The method is to copy the entire string onto the stack, back up the input head to the beginning, and begin checking for mismatches. If none are found and the stack is emptied, an initial palindrome has been found. If a mismatch occurs, the initial matching portion of the input is used to recopy the stack. Then the stack is popped

Author's address: Computer Center and Department of Information Engineering, University of Illinois (Chicago Circle), Chicago, IL 60680.

[1] Our model of a random access machine (RAM) is Cook's [2]; the reader is referred to the original paper for a careful discussion. Only two salient points need to be discussed here. Cook calls an algorithm *linear-time* if, given input of length n, the time required by the algorithm is of order $O(n * l(n))$, where $l(n)$ is the time needed to access a number of order n. Cook argues that a reasonable model of an RAM is one for which $l(n) = O(1)$. He then goes on to argue that once this is granted, it is consistent to postulate that $a(n) = c(n) = O(1)$, where $a(n)$ is the time required to add, and $c(n)$ the time required to compare two numbers of order n. This assertion is crucial to our scheme, in which addition, subtraction, and comparison are performed extensively.

one and the process is repeated. A similar mechanism can easily be found to recognize $S = \{wcx\}$, where w and x are nonvoid strings over an alphabet not including the letter c, and w is a substring of x. From this comes a linear-time "pattern-matching" algorithm which is not self-evident. This algorithm was first reported by Knuth et al. [5] and later given an appealing presentation by Hopcroft et al. [4]. The language $P = (ww^R)^*$, with w nonvoid, is known to have a linear-time recognition algorithm, discovered by Pratt. Pratt's algorithm is unpublished, but mentioned in [5]. Cook's theorem is used as a lemma. Pratt proves the following additional lemma: If T is a string of the form $(ww^R)^*$, and $T = T'T''$, and if T' is of the form $(ww^R)^*$, then so is T''. Using this lemma, Pratt finds T' by looking for an initial even palindrome (not necessarily the smallest) by the following device: The first two letters of the input are first considered and examined by Chester's method, mentioned earlier. If these fail to yield an even palindrome, four characters are examined, then eight, and so forth. Every time an initial palindrome is found, it is "discarded" and the remainder of the input string is subjected to the algorithm. This algorithm is not simply a simulation of a two-way DPDA. In fact, no one has ever found a two-way DPDA that recognizes P. The present author conjectures that one does not exist.

These algorithms, it must be clear, shed no light on the problem posed in the abstract. Two-way DPDAs seem to have the property that it is easy to test a string for a given property Π by examining successively shorter strings, but very difficult if not impossible to test for Π by examining successively longer strings. Intuitively, this is the reason we doubt that two-way DPDAs are strong enough to shed light, even as lemmas, on the problem of finding the initial shortest palindrome in time linear in the length of the palindrome, *without considering any letters not in the palindrome.* Nor can Weiner's algorithm [7] be of direct help, because it examines the entire input string.

The present algorithm has some of the flavor of algorithms resulting from the simulation of two-way DPDAs in that, as in [5], information already obtained from unsuccessful attempts to prove that a smaller initial substring was an even palindrome is kept for further use. The algorithm maintains an auxiliary one-dimensional integer array M whose length is one less than the length N of the input string. (If the input length is unknown at the beginning, the algorithm will still work if M is understood to be semi-infinite. The number of words of M actually used will still be $N - 1$). The ith cell of M represents an "interstice" between the ith and $(i + 1)$-th symbol of the input string. The number stored in $M(i)$ is the number of symbols in the input string mirrored about this interstitial position. Thus, for input $y = 0100110$, the successive values of $M(i)$ computed in the course of the algorithm will be 0, 0, 2, 0, 2, 0.

In order to visualize the operation of the algorithm, we will exhibit only those "special" positions i that lie in the middle of an even "group," i.e. a stretch of identical symbols. Such positions are necessarily the only ones that can bisect an even palindrome. Thus, while the algorithm in fact considers every position of y, we will display only positions 3 and 5. These are displayed as blocks linked to adjoining blocks. The lines linking blocks represent intervening "nonspecial" positions that are not exhibited. The algorithm uses a *cursor* to scan the input string left to right, maintaining at all times a record of where the *tentative center* (TC) of the initial palindrome resides. At the beginning, there is no TC. As the algorithm progresses, the TC gradually moves left to right, but more slowly than the cursor; the cause of its moving forward is its failure to lie at the center of an initial palindrome.

To show how the algorithm works, the following illustration should suffice. Let the input string $x = 0100110000110011000011001001101 0$. Let c be the cursor position. The algorithm being iterative, we show just one step. Suppose $c = 12$. The even groups thus far discovered are the 00 at positions 3 and 4, the 11 at 5 and 6, the 0000 at 7 through 10, and the 11 at 11 and 12. (The 00 at 13 and 14 has not yet been "discovered," because positions 13 and 14 are beyond the cursor.)

The structure built so far by the algorithm for purposes of its internal calculation is

shown in Figure 1. By a process that will become clear presently, suppose that we have already chosen the four zeros at positions 7 through 10 at the TC. Every new input letter is compared with the letter on the opposite side of the TC. If a match occurs, the next letter is considered. If matches continue to the beginning of the string, the TC is the center of the shortest initial palindrome. Clearly the most difficult part of the algorithm is its handling of mismatches.

To show what we do in the case of a mismatch, return to our example. The letters at positions 9 through 12 match their duals on the other side of the TC. Now advance the cursor to position 13. Again a match is found. Now set $c = 14$. Again a match is found. Setting c to 15, we once again obtain a match. Moreover, since letter 15 is different from letter 14, we have a new group of zeros, those at positions 13 and 14. This group is even, so it is added to the even groups, which now have the form shown in Figure 2. Advancing the cursor to 16, we obtain a matching failure; the letter at position 16 fails to match letter 1, its symmetric dual about the TC. Now we must determine which even group to choose as a new TC. Clearly, the algorithm will be linear if both (1) the array M needs to be scanned only to the right of the current TC and (2) the amount of work done at each position of M is constant, because the algorithm in effect scans the input once and each cell of M once, performing only a constant amount of work at each step. Our algorithm satisfies requirement (1) because it guarantees that all the even groups to the *left* of the TC have already been rejected as possible leftmost-palindrome centers. We now show how requirement (2) may be satisfied. The essential idea is to associate with each even group that has "failed," i.e. is not the center of a leftmost palindrome, an integer "failure number," that is, the number of letters to its right that match the letters to its left. This number is inserted as soon as it is discovered that the node has failed. The situation can be visualized by continuing our example after moving the cursor to 16 and discovering a failure. The situation is shown in Figure 3, in which the list has been "bent" in order to show the TC as a "vertex."

In Figure 3, the inverted triangle means "beginning of input string" and the double line indicates a matching failure. The number m, as shown, is the failure number, $m(i)$. Upon failure, the algorithm enters m into a field in the node representing the TC. In the present example, the TC has just failed, and its failure number has been entered into it. Since the nodes to its left have also failed (i.e. nodes α and β in Figure 3), they also contain failure numbers. We now consider the nodes to the right of TC in left-to-right order. (It is clear that each such node must have a symmetric image about the just rejected TC, i.e. α for α' and β for β' in Figure 3.) Define an integer n for each such node, defined as the number of letters just beyond it up to the double line. In our example, the first such node is α', and $n_{\alpha'} = 3$, indicating the length of the substring commencing at position 13 and ending

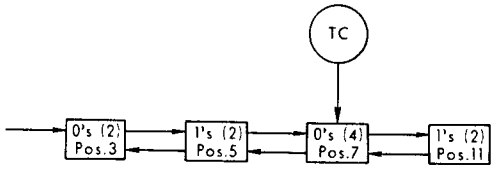

FIG. 1

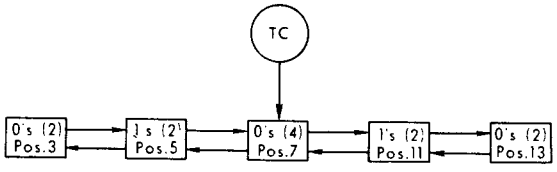

FIG. 2

at 15. The critical part of the algorithm is to compare $n_{\alpha'}$ with m_α, the failure number for α. Three cases may occur.

Case 1. $n_{\alpha'} > m_\alpha$. (See Figure 4.) In this case, the initial matching strings of length m_α, together with the nonmatching letters on either side, are both faithfully reproduced around α', by symmetry about the just rejected TC and because $n_{\alpha'} > m_\alpha$. Consequently, α' cannot be a TC and can be summarily rejected. In this case it is clear, moreover, that $m_{\alpha'} = m_\alpha$; the assignment $m_{\alpha'} \leftarrow m_\alpha$ is now made.

Case 2. $n_{\alpha'} < m_\alpha$. (See Figure 5.) In this case, it is known by symmetry about the just rejected TC that the initial $n_{\alpha'}$ letters on either side of α' match. The fact that a mismatch occurs just beyond (see Figure 5) must mean that α' is not a TC and that, moreover, $m_{\alpha'} = n_{\alpha'}$.

Proof. If the next letter had not caused a mismatch about the just rejected TC, then by symmetry the initial $n_{\alpha'} + 1$ letters on either side of α' would have matched. The fact that a mismatch did occur indicates that a different letter was chosen, and this must therefore produce a mismatch of letters just beyond the initial $n_{\alpha'}$ letters. □

In cases 1 and 2, our algorithm rejects α' and passes on to β', assigning $\min(m_\alpha, n_{\alpha'})$ to $m_{\alpha'}$. If β' fails, it passes on to the node beyond β', etc.

Case 3. $n_{\alpha'} = m_\alpha$. In this case, it is possible for the new node to be the TC. It then becomes the TC, with the assurance that all nodes to the left have been rejected and have been assigned a failure number.

In the present example, α' will be summarily rejected because $n_{\alpha'} = 3$ while $m_\alpha = 2$; its failure number will be set to 2. β' will not be rejected because $n_{\beta'} = 2$ and $m_\beta = 2$; in fact, it turns out to be the center of the initial palindrome.

The algorithm is encoded in ALGOL. D is the input string, and its symbols are represented as integers.

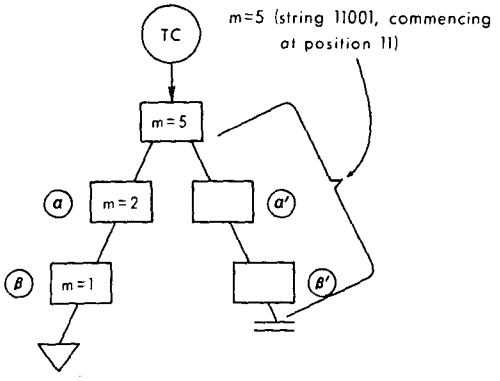

FIG. 3

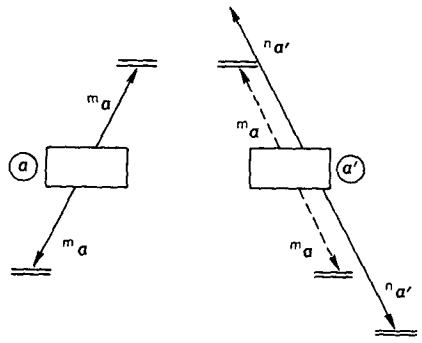

FIG. 4

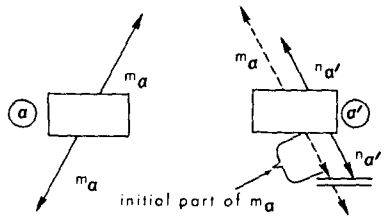

FIG. 5

```
INTEGER PROCEDURE PAL(D, N); VALUE N; INTEGER N; INTEGER ARRAY D;
BEGIN INTEGER COUNT, ENP, MDP, BP; INTEGER ARRAY M(1:N - 1);
COMMENT N is the length of the input string. ENP (end pointer) is the cursor, MDP (middle
  pointer) is the temporary center, and BP (beginning pointer) is the element of D that is the mirror
  image of the cursor about the TC. All of these of course are indexes. The procedure PAL returns
  either the index just beyond the initial even palindrome, if there is one, or else 0 if there is none;
ENP := 1; PAL := 0;
L1: ENP := ENP + 1; IF ENP = N + 1 THEN GOTO DONE;
MDP := BP := ENP - 1; COUNT := 0;
L2: WHILE D(ENP) = D(BP) DO
  BEGIN
  COUNT := COUNT + 1; ENP := ENP + 1; BP := BP - 1;
  IF BP = 0 THEN
    BEGIN
    PAL := ENP; GOTO DONE
    END;
  IF ENP = N + 1 THEN GOTO DONE
  END;
M(MDP) := COUNT; COMMENT Filling in # of symbols mirrored about MDP;
FOR F := 1 STEP 1 UNTIL COUNT DO
IF M(MDP - F) ¬= COUNT - F THEN
M(MDP + F) := MIN(COUNT - F, M(MDP - F)) ELSE
  BEGIN
  MDP := MDP + F; COUNT := COUNT - F;
  BP := MDP - COUNT; GOTO L2
  END;
GOTO L1;
DONE:
END
```

The extensions of this algorithm to other closely related problems are straightforward.

To find the initial odd palindrome of length greater than or equal to 3 (the general case of length 1 or greater being trivial), associate the cells of M with the symbol positions of D, rather than the interstices, and disallow "palindrome" solutions with $m = 0$.

To find the initial even or odd palindrome of length greater than or equal to 2, combine the original algorithm with the above variant, running the two "in parallel."

To test whether the original string is of the form $(ww^R)^*$, obeying the stated constraints of the original algorithm, apply the algorithm repeatedly, removing the leftmost palindrome found on the previous iteration. Stop when only the void string is left. (Pratt's lemma, mentioned at the beginning of this paper, assures the success of this scheme.)

Finally, we mention a few general points.

(1) We have called our algorithm "on-line" because it does not examine any symbols beyond the initial string it is looking for. Fischer [3] has posed the problem of finding *all* the prefix palindromes as follows. Given a string $X = X_1X_2\cdots X_n$, compute the Boolean vector $Z = Z_1Z_2\cdots Z_n$, where $Z_{i+1} = 1$ if $X_1X_2\cdots X_i = X_iX_{i-1}\cdots X_1$, and 0 otherwise. Thus Z_{i+1} is 1 if i is the index of the end of an initial palindrome and 0 otherwise. It is clear that a small adaptation of our algorithm will compute this function on-line in linear time. The adaptation consists of (a) inserting 0 into Z for those positions of the cursor where a palindrome is not discovered, (b) inserting 1 into Z for the positions in which a palindrome is discovered, and then proceeding as if a failure had occurred. (The value of m inserted into the TC is of course half the length of the discovered palindrome.) Combining the algorithm, thus adapted, with its odd-length analogue, as explained in the last paragraph, solves this problem.

(2) It appears possible that the ideas described can be adapted or extended to provide *real-time* recognition of $(ww^R)^*$ on an RAM, where by real-time we mean that there is one left-to-right scan of the input and a bounded number of legal operations (in Cook's

sense) between examinations of each symbol and between examination of the last symbol and issuance of a yes-or-no answer. This is perhaps not altogether surprising, as a Russian author [6] claims to have proven an even stronger result, namely, that one may examine a string with a multitape Turing machine that will determine in real time, upon reaching the ith symbol, whether that symbol is the end of the shortest initial even palindrome.[2]

(3) The remarks of the two preceding paragraphs indicate that reasonably defined on-line and real-time RAMs are computational models deserving serious study; we believe that their properties are nontrivial and will prove not to be analogous to results obtained for on-line and real-time Turing machines.

(4) The language $(ww^R)^*$ may be of interest in another connection. It is an open question [1] whether two-way DPDA languages include context-free languages. We feel strongly that the answer is no, and that a good way to prove it is to show that $(ww^R)^*$ is not a two-way DPDA language. If that can be proven, it follows that two-way DPDA languages do not include on-line-recognizable languages either.

ACKNOWLEDGMENTS. The author would like to thank the referee for his sharp and timely comments, which materially aided the preparation of this paper. Dr. Alfred Aho also provided useful guidance.

REFERENCES

1. AHO, A. *Currents in the Theory of Computing*. Prentice-Hall, Englewood Cliffs, N. J., 1973.
2. COOK, S. A. Linear time simulation of deterministic two-way pushdown automata. *Information Processing 71*, North-Holland Pub. Co., Amsterdam, 1972, pp. 75–80.
3. FISCHER, M. String matching and other products. Project MAC Memo. 41, M.I.T., Cambridge, Mass., Jan. 1974.
4. HOPCROFT, J., AHO, A., AND ULLMAN, J. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, Reading, Mass., 1974.
5. KNUTH, D. E., MORRIS, J. H., AND PRATT, V. R. Fast pattern matching in strings. Tech. Rep. CS 440, Computer Sci. Dep., Stanford U., Stanford, Calif., 1974.
6. SLISENKO, A. O. Recognition of palindromes by multihead Turing machines. Proc. of the Steklov Math. Inst., Acad. of Sciences of the USSR, Vol. 129, 1973, pp. 30–202.
7. WEINER, P. Linear pattern matching algorithms. IEEE Symp. on Switching and Automata Theory, Vol. 14, 1973, pp. 1–11.

RECEIVED DECEMBER 1973; REVISED NOVEMBER 1974

[2] This paper, of some 200 pages, is still untranslated and the proof apparently in some doubt. The best reliable result on an on-line Turing machine is due to Fischer [3]; it requires time $n \log n$, where n is the length of the input string.