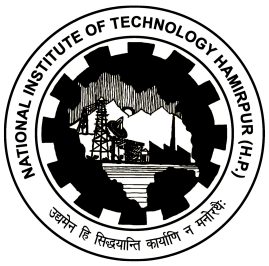
Real-Time Application of Multi Track Turing Machine

Theory Of Computation
CSD-225



Submitted by:

Kashish Srivastava (185014)
Dipesh Kumar (185015)
Akash Rana (185034)

CSE (4 Year) : 4th Semester

Under the guidance of

Dr. Nagendra Pratap Singh
Assistant Professor, CSE Department
National Institute of Technology, Hamirpur

Department of Computer Science and Engineering
National Institute of Technology, Hamirpur

Abstract

A Turing machine is an abstract mathematical model of a computer. Recall that a Turing machinecan move back and forth in the working tape while reading and/or writing.There are several variants of Turing machines in the literature. Our variant of a Turing machineconsists of a finite-state control (also called the program), a read-only input tape where the inputis given and a working tape (also called memory) where computations are performed.In computing an answer, intermediate results are computed and kept in the working tape sothat it can be referenced later for further computations.

A real-time Turing machine algorithm that finds the smallest nontrivial initial palindrome in the input string is constructed. A small modification of this algorithm yields a real-time Turing machine algorithm which finds all initial palindromes in the input string.

Contents

Palindrome detection in real time

An Application of multi track Turing machine

Palindromes are sentences that read the same backwards as forwards. Here are a few examples.

A man, a plan, a canal, Panama.

Dennis and Edna sinned.

Red rum, sir, is murder.

Live not on evil, madam, live not on evil.

Able was I ere 1 saw Elba.

Was it a rat I saw ?

No lemons, no melon.

Sums are not set as a test on Erasmus.

These and many more can be found in [3]. A two-dimensional (5 x 5) eight-hundredyear-old palindrome (it also reads the same upwards and downwards) in Hebrew will be supplied by the author upon request.

Formally, given an alphabet $\sum$, the set P = {x $|x\epsilon\sum^+, x = x^R\}$ is the language which consists of all the palindromes over $\sum$. (x^R is the string x reversed.) If we restrict our attention to even (length) palindromes we get the language P' = $\{x|x = ww^R, w\epsilon\sum^+\}$. Note that P and P' are basically the same recognition problem. Any algorithm that recognizes P can be changed to one that recognizes P' by intersecting with a regular set. Conversely, any algorithm that recognizes P' can be changed to one which recognizes P by considering each symbol twice. The language P' is a favorite example in language theory. For example, P' is a contextfree language (cfl) which is not deterministic [12]. (The close relative, the set $\{x|x = wcw^R, w\epsilon\sum^+\}$, of P' is a deterministic cfl.)

The language P' is also a favorite example in the theory of computational complexity. Perhaps due to the very simple definition of P', it became a challenge to researchers to construct efficient algorithms to recognize P' on various computing models from the very restricted model of the one-tape

Turing machine to the very powerful iterative array model. It is known that a one-tape Turing machine whose working tape is the input tape can accept P' in time $O(n^2)$; and every such machine that accepts P' runs at least cn^2 steps on infinitely many inputs. Also, it is known that a one-tape Turing machine whose working tape is not the input tape can accept P' using O(log n) space; and every such machine that accepts P' must use c log n space on infinitely many inputs [12].

On the other hand, it was quite surprising when Cole showed in 1964 how to recognize P' in real time on an iterative array [4]. (See also [5].) Very recently Seiferas [16] defined an iterative array with central control and showed that it can be converted to a conventional iterative array with the same time behavior. Then he gave a five-instruction program for such a machine which recognizes P in real time. Both algorithms heavily use the ability of the iterative array to consider all characters of the input simultaneously. Therefore, neither algorithm implies any of the fast algorithms for recognizing P mentioned below. Actually it is not known whether an iterative array can be simulated by a random access machine with no time loss. For example, an iterative array can multiply two n-bit numbers in real time [2], while no linear-time multiplication algorithm is known for a random access machine.

We now turn to the more popular computing models, the random access machine with unit cost (RAM) and the multitape Turing machine (Tm). Obviously, an off-line Tm can recognize P in time 2n. The more interesting problems are:

Problem 1. 'To recognize the language $\{ww^Ru|w, u\epsilon \sum^+\}$;

ProbEem 2. To find all initial palindromes in a given string.

These problems turn out to be closely related to the string-matching problem: Given X, y $\epsilon \sum^+$, find the first (or all) occurrences of x in y. The discovery of the existence of a linear-time algorithm on a RAM for Problem 1 led to the discovery of a linear-time string-matching algorithm by Knuth et al., [13]. They also indicated how to solve Problem 2 in linear time on a RAM. Following [13], Fischer and Paterson showed how to do string-matching in linear time on a Tm [8]. Th e y used it to construct a linear-time algorithm for a Tm for Problem 2 (and hence also for Problem 1). Although their stringmatching algorithm is on line, the algorithm for Problem 2 is inherently off line: it matches x versus x^R so it has to see the complete string first. They also indicated how to derive an O(n log n) on-line Tm algorithm for Problem 2. By on line we mean that the machine identifies the initial palindrome before it reads the symbols following it. In

an attempt to derive faster algorithms the following questions come to mind.

Question a. Can we recognize palindromes on line in linear time ?

Question b. Can we do it in real time ?

Question c. Can we do string matching in real time ?

By real time we mean that the machine is on line and in addition it makes only a constant number of steps between two readings. (One step in the case of a Tm.) For formal definitions of on line and real time see Section 2.

Manacher [14] presented an on-line linear-time RAM algorithm for palindrome recognition, settling Question a affirmatively in the case of the RAM. (It is interesting to note that the existence of such an algorithm could be derived using several theoretical results on fast simulations [9].) Manacher also conjectured that a real-time algorithm exists. In [lo] we defined the predictability condition, a sufficient condition for an on-line algorithm to be transformable into a real-time one. Manacher's algorithm satisfies the condition. Hence his conjecture follows immediately. In a similar way, various stringmatching algorithms (for RAM and Tm) either satisfy the condition, or can be modified so that the condition holds. Thus, the predictability condition allowed us to settle affirmatively Question c [lo].

In this paper we settle affirmatively the remaining questions, Questions a and 6 for Tm. We construct a real-time algorithm for Tm that finds the smallest nontrivial initial palindrome in the input string. (Nontriviality excludes the smallest initial palindrome which always consists of the first input symbol.) We then indicate how to modify it so that it finds (in real time) all initial palindromes in the input string.

These results are not completely original. We know of an existing paper by Slisenko that solves the same problems l-171. Daley [6] has translated it into English. Using Slisenko's basic notion-the chain, and 4 of his 12 lemmas on strings, we were able to come up with a much shorter algorithm. At the very least we now have a much shorter and more comprehensible construction and proof. (Slisenko's paper is 173 pages long.)

The structure of the paper is as follows. In Section 2 we review some preyious definitions and results needed for the construction of the algorithm. In Section 3 we describe some properties of strings. All the concepts and most of the results in Section 3 are due to Slisenko. (All the other sections are original.) Section 4 is a sketch of the algorithm. Sections 5-7 describe in detail the major parts of the algorithm.

Sketch Of algorithm

The algorithm will have a tentative center C (a head) and two heads L and R which move left and right, respectively, comparing symbols. If L hits the left endmarker a palindrome is found. The tentative center satisfies inductively the property that no place to its left can be the center of the initial palindrome that is currently being sought. (Note that in our case no place to the left of C can be the center of the smallest (nontrivial) initial palindrome. We used the definition above, since we shall show later how to modify the algorithm so that it finds all initial palindromes. In that case there can be some places to the left of C that are centers of initial palindromes. But these palindromes have already been found.) This process of matching symbols is performed by the procedure match defined below. Whenever match is called the string in (L, R) is a palindrome.

procedure match

$while a_L = a_R do[L \leftarrow L - l, R \rightarrow R + l]$

if $a_L = \bar{a}_R (i.e.$, they match but $L = 1)$

then STOP-a palindrome has been found.

else STOP-a mismatch has occurred.

end

Match is the heart of the algorithm. One form of it or another will run in parallel with the other procedures. R is the reading head. It reads input symbols only in odd places. In even places R "reads" the special symbol. Before moving from an odd place to the next (even) place it prints 0 unless a palindrome is found. L, R, and C will refer either to the places or to the heads which always scan the corresponding places. No confusion will arise. Note that since L is decreased by 1 and R is immediately increased by 1, R - L will be always even when a_R is compared to a_L. Hence either R and L are both even and $a_R = a_L$. =the special symbol or both are input symbols. So, when a mismatch occurs both Us and a, must be input symbols. The algorithm will spend more than a constant amount of time only when a mismatch occurs. Let l = (R + 1)/2, z = $b_1 ... b_{l-1}$, and $a = a_R = b_l I$. As a result of the mismatch C will move right until it finds the next tentative center. It will always be the case that if C moves s places to the right dt(z, a) = O(s). But dk(z, a) = 2s - 1 (see Fig. 1). So the predictability condition will hold and our algorithm can be made real time.

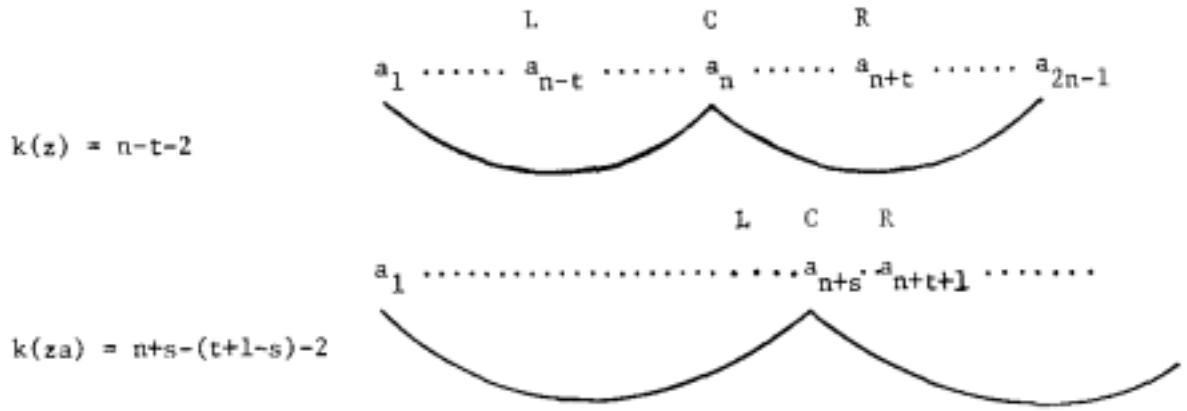


Figure 2.1: Before and after the mismatch.

In the sequel we always assume that the palindrome which is sought has not been found, since otherwise we are done. Hence if match stops, then a mismatch has occurred.

Assume a mismatch occurs. Let k = R - C and let C' be the center of the longest initial palindrome in [L, RIR (which must be the next tentative center, since nothing in between C and C' can be the center of an initial palindrome).

We distinguish between two cases: (1) $C' - C \leq k/4$ or the chain case, and (2) $C' - C > k/4$ or the nonchain case.

The following claim explains the nomenclature used above.

Claim 1. The chain case holds and D = C' if and only if there exists a chain CH such that

(a) C and D are adjacent nodes of CH,

(b) CH contains at least three nodes to the left and at least three nodes to the right of C, between L and R,

(c) $R \leq CH^+$.

The case above will be referred to as the chain cake w.r.t. CH and C. Until a mismatch occurs we have enough time to discover the chain CH, if it exists, at least the part left of C. This is done in parallel to matching L and R. If such a chain is found the algorithm will take advantage of the symmetries of the chain in order to proceed. Otherwise the nonchain case holds. Using the FPP (the linear-time off-line procedure which finds all intial

palindromes in a given string [8]), we find the largest initial palindrome in $[L, R]^R$. (Note that a palindrome of size 1 always exists.) Let C' and L' be the center and the left end of this palindrome. We set C to C' and L to L' and continue. The time spent before reading the next input is O(k). But dk(z, a) $\geq$, k/2 - 1 as was explained above, since the tentative center moves at least k/4 places to the right. So the predictability condition holds.

Proof of Claim 1. Assume the chain case holds. So c' - C $\leq$k/4. It is easy to see that C k-absorbs C'. By Corollary 1 and since C' - C $\leq$ k/4, there is a chain CH such that C, C' ϵ CH and (b) holds. C' and C are adjacent nodes of CH since otherwise if there was another node, C", in between them, then there would be a larger initial palindrome in $[L, R]^R$ with C" as its center. So (a) holds. Since R(C')> K - (C' - C), the second part of Corollary 1 implies (c).

Now assume that there is a chain CH that satisfies (a)-(c). Obviously D is a center of an initial palindrome in $[L, R]^R$. By Lemma 3 it is the center of the largest one and thus D = C'. By (b) C' - C $\leq$ k/4 and the chain case holds.

Figure 2 contains a flow diagram of the algorithm, and Fig. 3 describes different stages in discovering and maintaining the chain. Both figures are intended to help the reader in reading the following sections.

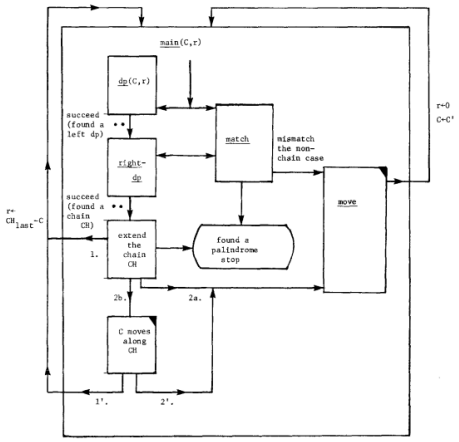


Figure 2.2: A flow diagram for main(C, r). In fact main1 is called from move and not main but main1 is similar to main. (The marked boxes cannot occur in main1 .)

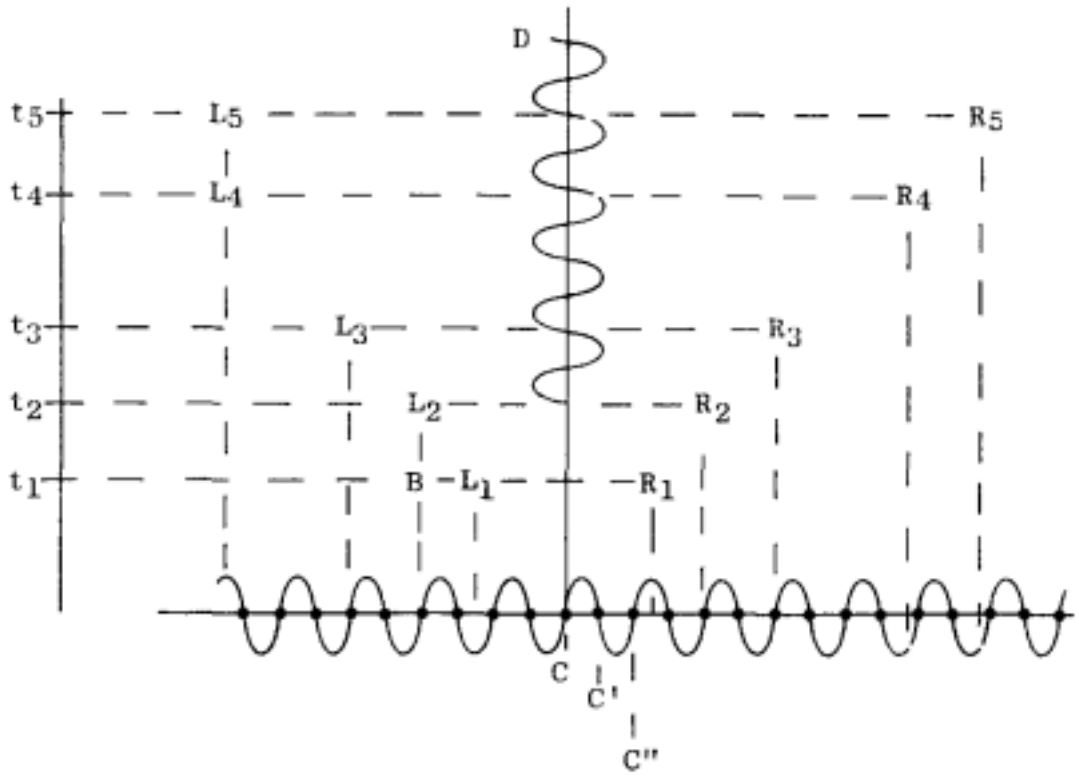


Figure 2.3: Discovering and maintaining the chain: t_1 , a dp is found; t_2, right dp succeeds; t_3, after extending the chain in one period; t_4 , $a_D = a_R \neq a_L$, C moves to c'; t_5 , after one iteration of steps 1 and 2, C moves to C"

Finding the Chain

The main procedure is main(C, r). Its initial function is to find a candidate chain for the chain case. Whenever it is called the conditions for main(C, r) hold; namely, (1) r $\leq$ R - C$\leq$ 5r/3; (2) if there is a chain CH in which C is an interior node and h(CH) $\leq$ r, then $CH^+ <$ R; and (3) the string in places [L, R] is a palindrome. Condition (2) and Claim 1 (c) imply that if the conditions for main(C, I) hold and the chain case will later hold w.r.t. C and CH, then h(CH)>r.

Main(C, r) will first look for a chain through C with step>r and with at least three places to the left and three places to the right of C. To describe m and (C, r) we need a definition. We assume below that the conditions for main(C, r) hold.

A double palindrome (dp) is a palindrome of length 4K + 1, k> 0, such that its first (and hence its last) 2K + 1 symbols form a palindrome. k is called the step of the dp.

EXAMPLE.

$$aw^R\ bwaw^R\ bwa$$

$$1 \qquad 2 \qquad 3$$

The string above is a dp. Note that there is a one-to-one correspondence between dp's and ch.g.'s with three places. (The step of the dp and the step of the corresponding ch.g. are the same.)

Claim 2. Assume CH is a chain through C with h(CH) = h > r and with at least three nodes to the left and three nodes to the right of C. Suppose CH is the chain with the smallest step among such chains. Then the substring in places [C - 4h, C] is the smallest dp with step h > r that ends at C.

Proof. The string in places [C - 4h, C] is obviously a dp. Assume the smallest dp which ends at C and with step h'>r satisfies h'<h. So there is a ch.g. through S = {C - 3h', C - 2h', C - h'}. Hence, there is a chain CH" through S with step h" which divides h'. Since C is 4h-symmetric, CH" passes through C and has at least three nodes to the right and three nodes to the left of C. By the definition of CH h"$\leq$r. Note that C + 2h' ϵ CH". So R<C + 2r<C + 2h'< $CH"^+$, contradicting condition (2).

Claim 3. Suppose the smallest dp that ends at C and has step>r has

step h. Assume in addition that C is 4h-symmetric. Then the places C + ih, -3≤i≤3, are consecutive nodes of a chain.

Proof. Obviously these seven places form a ch.g. If they are not consecutive nodes of a chain, there is a chain with step h' through them (h' divides h). Since the string in places [C - 4h', C] is a dp, and h'<h, we have h'≤r. The contradiction then follows as in Claim 2.

Main(C, r) finds the chain with smallest step h>r with at least three nodes to the left and three nodes to the right of C by finding the smallest dp that ends at C with step>r. To find the latter it uses the procedure dp(C, r) defined below.

procedure dp(C, r)

Find the smallest dp which ends at C with step>r by applying stages i = 1, 2,..., until a STOP occurs. The r symbols left of C are marked when the call is made.

Stage i

Mark the symbols in positions j, C - $2^{i+2}r \leq j \leq$C. If you cannot (C - $2^{i+2}r <$l), record that this stage is final and mark only the available symbols. The marking is done by an extra head to double the length of the marked string. (For i = 1, it is doubled three times.)

Every doubling, mark with a special symbol the leftmost marked symbol.

Let u_i be the marked string.

Using the FPP find all initial palindromes in u_i^R and mark their left ends. (Their right end is always C.)

Use two heads to check if for r<k≤ $|u_i|$l/4 there are initial palindromes of size 2k + 1 and 4k + 1 in u_i^R.

If you find them (i.e., the pair with smallest K in this range) mark semiperiods (places C - k, C - 2k, C - 3k, C - 4k) and STOP Otherwise if the stage is final STOP.

end

Note that the i^{th} stage takes $O(|u_i|) = O(|u_{i-1}|)$ time units. (For convenience we define $u_0(u_{-1})$ to be the substring of size 4r + 1 (2r + 1) which ends at C.) The heads used for this procedure are called search heads. Main(C, r) runs match and dp(C, r) in parallel. During any stage of dp(C, r) match runs more slowly by waiting a constant amount of time between executing two consecutive steps. We choose the constant so that at the

beginning of the i^{th} stage of dp(C, r) $R - C \leq 5/12|u_{i-1}|$ and at its end R - C = $|u_{i-1}|/2$. Since the i^{th} stage takes O(l ui-r 1) time, we can choose the constant mentioned above so that R proceeds a distance$\leq |u_i - 1|/12$ during the i^{th} stage. So if $R - C \leq 5/12|u_{i-1}|$ at the beginning of the i^{th} stage, then R - C$\leq$ $|u_{i-1}|/2$ at its end. If at that point R - C$<$ $|u_{i-1}|/2$, dp waits for match until R - C =$|u_{i-1}|/2$, i.e., until L hits the end of the next ui (j = i - 2) which is marked. So just before the i + 1 stage $R - C = |u_{i-1}|/2 < 5/12|u_i|$ (if dp d oes not stop at the i^{th} stage). Initially $R - C \leq 5r/3 = 5/12|u_0|$. Hence this pair of conditions will hold inductively at the beginning and at the end of each stage.

If a dp is found in the i^{th} stage, then its size$>$ $|u_{i-1}|$. On the other hand when it is found, R - C$\leq$ $|u_{i-1}|/2$. Let B be the left end of the dp. So L - B$>$ $|u_{i-1}|/2$.

If a dp is found main calls the procedure right-dp. The latter checks whether there is a symmetric dp to the right of C. It applies match until L = B. (The Bth place is marked.) In parallel it rushes all search heads to C. Note that L - B$>$ $|u_{i-1}|/2$ and all the search heads are at distance $O(|u_{i-1}|)$from C. So again, slowing match down by a constant factor, all the search heads manage to reach C before L reaches B if a symmetric dp exists to the right of C.

Assume there is a chain CH through C with step h$>$r, and with at least three nodes to the left and three nodes to the right of C. Suppose CH is the chain with the smallest step among such chains. By the conclusion of Claim 2 the substring in places [C - 4h, C] is the smallest dp with step ¿r. Hence dp(C, r) will find this dp, and right-dp will find the symmetric dp to the right of C. So if a mismatch occurs either in right-dp or before dp(C, r) finds a dp, then the chain case cannot hold w.r.t. C and any chain CH through C: If h(CH)$<$r it cannot be the chain for the chain case since the conditions for main (C, r) hold, and by the discussion above there do not exist such chains through C with h(CH)$>$r. Hence in both cases the nonchain case must hold. As a result a call to move is executed. Move moves the tentative center to a new place C'$>$R + (R - C - 1)/4. It operates similarly to the description in the sketch of the algorithm. ,Note that if dp(C, r) stopped at the i^{th} stage R - C$\geq$ $|u_{i-2}|/2$, since the i - 1 stage was completed. So the distance of the search heads from C$<$ $|u_i| \leq$8(R - C). (This fact will be used later.) Move will be defined in the sequel. In the meantime we consider the case that right-dp has found a symmetrical dp to the right of C. By Claim 3 the places C + ih, -3$\leq$i$\leq$3, are consecutive nodes of a chain. We denote the chain by CH (h(CH) = h).

Extending the Chain

After these seven nodes of CH are found main finds the rest of CH as follows. (The need for keeping track of the chain information will become clear in the sequel.) It now runs a version of match which compares three symbols a_L, a_R, and a_D . D is an extra head (not a search head) which moves back and forth along a semiperiod next to C. Endpoints of semiperiods are marked on the way by L and R.

This stage continues as long as $a_D = a_L = a_R$. Note that if at some point $a_R \neq a_D$, then R = $CH^+ + 1$ and we shall say that CH^+ has been found. So this stage of extending the chain can terminate in one of the following cases.

(1) $a_L = a_R \neq a_D$,

(2) $a_L \neq a_R$, and

(2a) $a_D \neq a_R$,
(2b) $a_D = a_R$.

Consider Case (1). Since CH^+ was found, Claim l(c) implies that the chain case will not hold w.r.t. CH. We now show that there is still enough time to find a new candidate for the chain case: Let r = CH_{last} - C.

Claim 4. The conditions for main(C, T) hold.

Proof. Since R = $CH^+ + 1$, R - CH_{last} $\leq$2h. Also r$\geq$3h and hence r<R - C$\leq$5r/3. So condition (1) holds. Note that CH_{last} = C + r. By the first part of Lemma 4,any other chain $\hat{CH}$ in which C is an interior node and with h($\hat{CH}$) $\leq$r must satisfy h($\hat{CH}$) <h. So, by the second part of Lemma 4, any such chain CH satisfies $\hat{CH}^+ < CH^+$ = R - 1. Hence condition (2) holds. Obviously condition (3) holds too.

Hence, in Case (1) a call to main(C, I) is executed. Although the first r symbols are not marked, the place C - r is marked and can easily be identified. So dp(C, r) will be able to mark 8 places and to erase the old marks if its first stage is completed.

We now consider Case (2). In Case (2a), the nonchain case must hold. The chain case cannot hold w.r.t. and with h($\hat{CH}$)<, r, since and $\hat{CH}^+$ <R as in Claim4. (Recall Claim l(c).) It cannot hold w.r.t. and with h($\hat{CH}$) >r since h($\hat{CH}$)>r$\geq$(R - C)/2 and there cannot be three nodes of $\hat{CH}$ between

C and R. (Recall Claim l(b).) So a call to tlloete is executed. In Case (2b), the chain case holds w.r.t. C and CH. By Claim 1, c' = C + h, the next chain node. The following instructions are now executed.

Main now extends the chain but only to the right. D marks its place, L stays put, and two steps are now iterated.

(1) C moves to the next chain node, which is marked.

(2) D and R continue the match for the length of two additional semiperiods.

Except the procedure move described below, step (1) above is the only case in which dt(z, a) = h and cannot be bounded above by any constant, But C moves to the next chain node c' = C + h. Hence, as was explained above dk(z, a) = 2h - 1 and the predictability condition holds. The same argument holds each time D and R complete step (2) since at that point the current C is excluded from being the center of the initial palindrome and the next tentative center is C + h. (Again, this is so, since the chain case holds w.r.t. (the current value of) C and CH.) When a mismatch between a, and a_R occurs we distinguish between two cases:

(1') It occurs exactly at the end of (2) and a_L. = a_R . This case is analogous to Case (1) and m and (C, r) is called (r =CH_{last} - C). Note that in this case (only), one of the heads (D) is not at C when the call is made. But C - D<r. So D is rushed to C during the first stage of dp (which if completed takes > r time units). If the first stage stops in a mismatch and D has not reached C it is rushed to the new center C' which is found by move during the call to move. We shall later use the fact that when main(C, r) is called D>L.

(2') In all other cases we can conclude as in Case (2a) that the nonchain case holds. So, similarly a call mowe is executed.

The Procedure

To complete the construction we only have to describe the procedure move. It essentially operates as was described in the sketch of the algorithm for the nonchain case. In addition it does some extra work which was not mentioned there. But the predictability condition will still hold. To define move we need the off-line version of main(C, r) which we denote by mainl(C, r). Main1 will be explained later.

procedure move

Comment: Find a place c' for a new tentative center with C + (R - C - 1)/4 < c' ≤ R. Resume the match with the new center after cleaning the tape.

Erase all the old marks on the tape between L and R (not the symbols). Using the FPP, find the largest initial palindrome in $[L, R]^R$.

Let C' be its center.

Move search heads, C and D to C'.

Mark R as RR.

Move L and R heads to C.

Call main1(C, 0)

end

Move finds the position for the tentative center, c'. But before resuming the match it needs the chain information, i.e., whether there is a chain through C' with at least three nodes to its left and three nodes to its right, that in addition does not end before R (a candidate chain for the chain case). To find this information it calls mainl(C, 0) (C ←C before this call).

When mainl(C, r) is called R is not reading new symbols when it is increased by 1. Except the fact that mainl(C, r) operates off line it is exactly the same as main(C, r). Eventually R reaches RR (which is marked). At this point main1 switches to the corresponding point in main.

Consider the call to mainl(C, 0). Note that no mismatch occurs until R reaches RR. (C is (RR - C)-symmetric.) So before RR is reached if a chain with seven nodes is found the chain is extended. Only Case (1) can occur before reaching RR. (Cases (2a) and (2b) correspond to a mismatch.) In this case mainl(C, r) will becalled with the corresponding r. Hence R reaches RR in one of the following:

(a) In the middle of dp(C, r)

(b) In the middle of right-dp

(c) In the "extending the chain" stage.

In all cases the corresponding procedure continues as though it was called from main(C, r). Note that the time it takes R to reach RR is O(RR - C), since main(C, r) and mainl(C, r) have the same run time and in the absence of a mismatch main is real time. In all the cases where move is called C' - C > (R - C - 1)/4. (In this paragraph L, R, and C refer to their value at the time of the call to move.) So dk(z, a) = 2(C' - C) - 1 $\geq$ (R - C)/2 - 1. In all these cases the distance of D and the search heads from c is O(R - C). (L < $D \leq$C and the search heads are at distance O(R - C) if move was called due to a mismatch at dp or right-dp.) So it takes O(R - C) time to move these heads to C'. Also it takes O(R - C) time to find C' by the FPP and to clean the tape between L and R. It takes O(RR - C') = O(R - C) time for m and zl(C, 0) to run off line until R reaches RR and main regains control. So the total time spent dt (z, u) = O(R-C) and the predictability condition holds.

The complete algorithm starts as follows:

read b, into the input tape, mark it and output 0

C, R, L, D, search heads $\leftarrow$ 2 (to exclude a palindrome of size 1)

call main(C, 0)

Conclusion

We constructed a real-time algorithm for a multitape Turing machine for finding the smallest (nontrivial) initial palindrome. It can be easily changed into a real-time algorithm which finds all initial palindromes. Except in one subcase, when an initial palindrome is found the machine prints 1 and then behaves as though a mismatch occurred and thus looks for the next initial palindrome. The only exception is in the case when an initial palindrome is found while having a chain CH and the chain case holds (Case (2b)). In this case the machine behaves as in Case (2b). It extends the chain to the right. Recall that (1) C moves to the next chain node, and then (2) D and R compare symbols along a string of length 2h. Each time step (2) is completed a new initial palindrome is found. Similarly, there is a real-time algorithm which finds the smallest (all) even (odd) initial palindrome(s).

Let L = $\{ww^r\}^*$. Pratt has observed that if $vv^Ru\epsilon L$, then $u\epsilon L$[13]. So the real-time algorithm which finds the smallest initial even palindrome can be easily transformed into a real-time algorithm which recognizes L. The latter uses the former as a subroutine which is called whenever an initial even palindrome is found. This obviously works in the case of a RAM. It works also in the case of a Turing machine. Recall that the real-time multihead Tm described above is converted into a real-time multitape Tm. So each head starts at the leftmost cell of the corresponding tape. Whenever an even palindrome is found each head marks the cell it scans. The marks will serve as left endmarkers of the corresponding tapes. So the search for the next even palindrome starts over with the same initial conditions.