Lower Bounds on String-Matching

Ming Li*

TR 84-636
September 1984

Department of Computer Science
Cornell University
Ithaca, New York 14853

*This work was supported in part by an NSF grant MCS-8301766.

Lower bounds on string-matching

Ming Li*
Department of Computer Science
Cornell University
Ithaca, New York 14853

Abstract

New techniques for obtaining lower bounds on string-matching problems are developed and we prove the following new results.

String-matching cannot be performed by a three-head one-way deterministic finite automaton. This answers the $k=3$ case of the open question, due to Galil and Seiferas [GS], whether a k-head one-way deterministic finite automaton can perform string-matching.

String-matching by a k-head two-way DFA with k-1 heads blind (can only see two end symbols) is studied, tight upper and lower bounds are provided.

Probabilistically moving a string on one tape (requiring n^2 time.) is harder than probabilistically matching two strings on 1 tape. Notice that this is not true for deterministic or even nondeterministic TMs. This is the first result showing that checking is easier than generating.

* This work was supported in part by an NSF grant MCS-8301766.

1. Introduction

The string-matching problem is defined as follows: given a character string x, called the *pattern* and a character string y, called the *text*, find all occurrences of x as a subword of y, as defined in [GS] and [LY]. It is well known that the string-matching problem is very important in practice.

Since the linear algorithms by [KMP] and [BM], there has been a constant effort to search for better algorithms to run in real time and save space. Finally, in [GS], Galil and Seiferas showed that string-matching can be performed by a six-head *two-way* deterministic finite automaton in linear time. They notice that a multi-head *one-way* deterministic finite automaton must operate in linear time. Motivated by this observation they ask whether a multi-head one-way deterministic finite automaton can perform string-matching. And in [LY], we negatively answered this question for the case of two heads. Efforts have been made on the k>2 cases, but even the k=3 case has not been solved. It is believed that a solution to the case of k=3 would give some important insights for the general case.

In this paper we develop new techniques and negatively settle the case of k=3. We hope the method used here combined with that of [LY] would help to provide useful techniques for the general problem.

Besides above, we also study string-matching by 2-way k-head DFA with k-1 heads blind in section 4, and probabilistic matching and moving strings on one Turing machine tape in section 5.

2. Preliminaries and Notation

We shall denote a k-head one-way deterministic finite automaton by k-DFA. A k-DFA M has a finite set Q of states, a subset A of Q of accepting states, k heads $h_1, h_2, \cdots, h_k$ (which may see each other), a transition function, and a one way read only input tape. We assume that the standard input to M, $\#pattern\$text\not c$ where $pattern, text \in \Sigma^*$ for the alphabet Σ={0,1}, and # and $\not c$, are the left and right endmarkers, respectively. Initially, all k heads are on the left-most tape position. At each step, depending on the current state and the ordered k-tuple of symbols seen by the heads $h_1, h_2, \cdots, h_k$, M changes state and moves some of the heads one position to the right. If during $|Q|+1$ consecutive steps no head is moved we know that no head will ever move again. We can modify M so that whenever this situation occurs, all heads move to the right until they reach the right end-marker. Hence we assume that on each input all heads will eventually reach the right end-marker, and M halts when this happens.

In [GS], it was not explicitly defined how should a k-DFA report all occurrences of the pattern; Here we will adopt the weakest assumption (which is the hardest for the lower bound proof.): we say k-DFA M performs string-matching correctly if,

> M is in some final states in A for exactly q steps iff the pattern occurs in the text for exactly q times.

Notice that we do not even require M to report a matching right after finding it. Concerning the Yes&No recognition version, we can also define L_k to be the language that *pattern* occurs in the *text* at least k times. We can show that L_k, say for $k>9$, is not recognizable by a 3-DFA. (We can do better than this, but with more effort.)

An *ID* of M *on* input m is the $k+2$ tuple: $(m, q, i_1, i_2, \ldots, i_k)$ where q is a state and i_j for $1 \leq j \leq k$ is the position of the j-th head. Clearly $1 \leq i_j \leq |m|$ for $1 \leq j \leq k$. The *initial ID* of M on input m is $(m, q_*, 1, 1, \cdots, 1)$. The *computation* of M on m is the sequence of ID's reached by M on input m, starting from the initial ID. A partial *ID* of M on input m at step t is $(m', q, i_1, \ldots, i_k)$, where m' is the suffix of m which is not seen by some head(s) of M yet.

Let I_1 and I_2 be ID's. We write $I_1 \vdash I_2$ if M, started in ID I_1, in one step reaches ID I_2. We write $I_1 \vdash_* I_2$ if $I_1 = I_2$ or if M, started in ID I_1, reaches ID I_2 in a finite number of steps. By k-NFA we shall denote the nondeterministic version of a k-DFA.

Let x and y be binary strings. We write $x = y$ if they are the same binary string. $|x|$ denotes the length of x. Superscripts will be used to denote the same string of different occurrences on the input tape. Subscripts will be used to denote different binary strings.

After first used by W. Paul [P], the Kolmogorov-Complexity has played an important role in the lower bound proofs. We define the Kolmogorov-complexity (K-complexity) of a string x, denoted $K(x)$, to be the length of the shortest program that prints x (only). A string x is *random* if $K(x) \geq |x|$. The conditional K-complexity of x w.r.t. y, denoted by $K(x|y)$, is the length of the shortest program which, with extra information y, prints x. We state two simple well-known facts without proof.

Fact 1: There exist random strings. As a matter of fact, most strings are random.

Fact 2: If string uvw is random, then $K(v|uw) \geq |v| - O(\log |uvw|)$.

In the following the word 'information' only means some binary strings. The 'amount of information' means the total length of the strings.

3. Three 1-way heads cannot do string-matching

Some more technical definitions and conventions are needed for string-matching.

For a 3-DFA M we will name the three heads h_a, h_b, h_c. We will also use h_1, h_2, and h_3 to mean the leading head, the second head, and the last head in a specific time, respectively. So h_a, h_b, and h_c are fixed names, whereas h_1, h_2, and h_3 are only transient names.

Let x be a string (a segment of the input of M) of length greater than 0. At a particular step in the simulation of M, we make following definitions. $p(h_i)=x$ denotes that the position of h_i is at the last bit of the x; $p(h_i)>x$ means that h_i passed the last bit of x; $p(h_i)<x$ means that h_i did not reach the first bit of x. The uniqueness of string x will be clear from the context, and if there are more than one possible x's, we always mean the last.

a_0 always stands for the *pattern* which is going to be of form $1^k X 1^{k'}$ for some X, k, k'. And X will be always equally partitioned into six parts $X = x_1 x_2 \dots x_6$. In general, given a string $x_{ij \dots k}$, without explicit definition, we will always implicitly assume that it is equally partitioned into six parts, and written as $x_{ij\dots k1} x_{ij\dots k2} \dots x_{ij\dots k6}$ with an extra same sized index. And when the ranges of the indices are not explicitly given, they are assumed to be from 1 to 6.

Let x and y be string segments on the input tape. We say x matches y if there is a time that one head, say h_i, of M is at x and another head, say h_j, is at y

simultaneously (excluding the first bit and last bit of x and y). If y is just another occurrence of x then we say this (occurrence of) x is matched, or matched by (h_i,h_j). We also say (h_i,h_j) did the matching. Let $x^1=x^2$ (which are just different occurrences of same string x at different places) be two segments on the input tape. We say x^1 is *matched_to* x^2 if there is a sequence of occurrences of x's starting from x^1, ending with x^2, each matches the next. An occurrence of x is *well-matched* if this occurrence of x is matched to the x of a_0.

The *text* in the input $\# 1^k X 1^{k'} \$ text ⊄$ is *easy* if: it can be constructed by X plus O(log $|X|$) more information, and there is a constant C_0 ($<<k,k'$) not depending on k, or k' such that each head position in the *text* can be described by $|k| + C_0$ information, and further if a head h is in an occurrence of X in the *text* then $p(h)$ can be specified by $C_0 log\,|X|$ information. As a matter of fact, the reader may simply assume that *text* will be consisted (almost) of blocks of $1^m X 1^m$'s (or $1^m X'$, where X' is some prefix of X) and $1^{m|Q|}$'s separated by single 0's. An *easy text* and a hard *pattern* is vital in our proof.

Although the proof of next theorem is tediously long and complicated, the idea behind it is simple. It came from one observation: Let kXk' be a K-random string. Suppose that there is a time that all three heads left the *pattern* $1^k X 1^{k'}$ and no head is reading ⊄ sign, the text is *easy*, and two heads are reading some occurrences of X, then we would lose the information of either k or k'. At this time we attach $1^l X 1^{l'}$ by the end of the *text*, if the machine does string-matching correctly, we would be able to recover k and k' by finding the minimum l and l' s.t. the machine finds one more matching. Therefore we show that kXk' is not random. So our goals are to (1) make *text* easy and (2) drive the heads out of *pattern* (or 1^k of *pattern*). To make *text* easy we construct *text* to be (almost) a sequence of a_i's and block of 1's, where $a_i=1^m X 1^m$ for some non-random m greater than k,k'; To drive the heads out of the *pattern*, we have to do an exhaustive adversary proof. We will try to construct 'bad' (but *easy*) inputs to fool the head out of *pattern*. Many cases have to be considered. But this is natural since the lower bound is basically a *universal* problem: *all* 3-DFA's do not do string-matching correctly. With the goal of 'constructing an *easier text* than the *pattern*' in mind, we start our proof. Drawing some pictures will be very very helpful in understanding the proofs.

Theorem 1 No 3-DFA can do string-matching correctly.

Proof : Suppose a 3-DFA M performs string-matching correctly, we will derive a contradiction. Fix a long enough Kolmogorov-random string Y. We will show that Y is not random for a contradiction. Divide Y=kXk', where $|k|=|k'|$ and $|X|^{1/4} >> |k|$, $|X| >> |X|^{1/2}$, and $|k| >> log\,|X|$. Let m=min$\{2^f \mid 2^f > k, k'\}$. X is divided into $x_1x_2x_3x_4x_5x_6$ of equal length as assumed above (and so is each sub-string). We will only consider inputs of form $\# 1^k X 1^{k'}$ $\$text¢$ to M. We will always denote the *pattern* $1^k X 1^{k'}$ to be a_0, which contains exactly information of Y. We will always assume that we are in the process of simulating M.

We need the following strategy **P** to play our adversary proof. The purpose of **P** is to either drive h_2 or h_3 out from certain area or to obtain an invariant value such that after h_1 passed a block of 1's, many more 1's can be added without changing the status of M. And this block of 1's can be used to recover k if it is followed by $X1^m$. For the easiness to understand, one may want to read **P** later when **P** is called.

P(x): Given $\#a_0\$text¢$ on tape, $p(h_1)=text$ with corresponding state of M and h_2,h_3 positions.

i:=1; append $b_i=1^{m|Q|}0$ to the input (before ¢);

while $S_1 \vee S_2 \vee S_3$ $\neg$*true do*;

i:=i+ 1;

append $b_i=1^{m|Q|}0$;

continue to simulate M until $p(h_1)=b_i0$;

od;

Where the three predicates are defined as below,

S_1: a matching of one occurrence of x (input of **P**) to some other occurrence of x by (h_2,h_3) happens in the last period of simulation (last while loop);

S_2: in the last period of simulation, (1) h_2,h_3 read only 1's, non of them moved $> |Q|$ steps; or (2) $h_2(h_3)$ read X or 0 and did not move, and $h_3(h_2)$ read only 1's and moved $< |Q|+1$ steps; or (3) both h_2 and h_3 read X or 0 at beginning, and they did not move.

S_3: h_1 and h_2 are separated by only blocks of 1's (separated by a single 0).

If S_2 is true, then there exist constants $C_1, C_2 < |Q| + 1$ such that for all l, replacing the last b_i by $a = 1^{C_1 + l * C_2}$ in the input, M is in a fixed state with same h_2, h_3 positions when $p(h_1) = a$. We replace the last appended $1^{m|Q|}0$ by $a_f = 1^{C_1 + l * C_2} X 1^m$ where $l = 1$ at this moment. If S_1 or S_3 is true, we do nothing.

end_P.

Remark 1: (1) Only one of S_i's can be true; (2) The number of times that the *while* loop to be executed is only $O(|X| + \#$ of 1 blocks in the input), assuming we have only less than C (a fixed constant not depending on the *pattern*) occurrences of X's in the *text*. Therefore after executing **P** for constant number of times, the *text* is still *easy*; (3) If S_2 is true then varying l does not affect the behavior of M.

Four lemmas are needed. Note: the a_i's used in each of the following lemmas are all 'local', that is, they have no relation with any a_i's used in the proof of other lemmas or of the main theorem.

Lemma 1: If the *text* is *easy* and x is any segment of X such that $|x| > |X|^{1/2}$, and no more than C occurrences of x exist in *text* for some constant $C << \log |k|$, then for x in each occurrence of the *pattern* (i.e. $1^k X 1^{k'}$) in *text*, x must be well-matched.

Note: There can be only one x occurring in X, or else X can not be random.

Proof of Lemma 1: Let x^l, for $l = 1, 2, ... l_0$, be all the occurrences of x which are not well-matched in the *text*. Now for each x^l, we record 3 pairs of information for 3 heads,

h_a pair: (positions of h_b and h_c and state of M when h_a first time at first bit of this occurrence of x^l, positions of h_b and h_c and state of M when h_a first time leave this x^l.);

h_b pair: exchange h_a and h_b in above;

h_c pair: exchange h_a and h_c in h_a's pair.

Now we show Y is not random. For input Y',

(1) Compare Y' with Y except the x part which we do not need.

(2) Construct the *pattern* and the *text* with x' of Y' (the corresponding part of x) replacing all x^l in above. Then for each of the above three pairs, starting from the first component, we simulate M until some ID of M coincides (matches) the second component of the pair. If there is no such coincidence we reject this Y'.

If Y' passed tests (1) and (2), then Y'=Y. And notice that the amount of information ($X-x$, $O(log\,|X\,|)$ information, h_a, h_b, h_c pairs for each x^l) we used in above program is less than $|Y|$ because of the assumption $|X\,|^{1/4} >> |k\,|$ and the fact $|x\,| > |X\,|^{1/2}$. □ (of Lemma 1)

Remark 2: Lemma 1 is true for a k-NFA, for any k. Combined with the ideas from [YR] the proof of a theorem of Yao and Rivest [YR] which says that k-DFA is better than $k-1$-DFA can ba be simplified.

The next lemma to be proved suggests the basic idea of the proof of our main result.

Lemma 2: If the *text* of an input is *easy*, and at some step two heads of M are in some (matching) X's in *text*, the other head's position can be described in $|k\,|$ long information and it is neither in 1^k of a_0 nor at the ¢ sign, then Y is not random.

Proof of Lemma 2: The partial current ID (current ID without the part that no head can see any more) at this time can be specified by the following short information of less than $|X\,| + 2(\,|k\,| + \,|k'\,|\,)/3$ long: X, $O(log\,|Y\,|)$ for constructing *text* and for specifying 2 matching head positions, $|k\,|$ for the third head position if it is not in a_0 or $|k'\,|$ if the third head is in a_0 (If it is in X of a_0 then specify $|k'\,|$ and need $log\,|X\,|$ for the position, if it is in k' of a_0 then only specify the distance from this head to the $ sign.). We then find the smallest i and j such that after appending $01^i X 1^j$ at the end of the input (before ¢ sign) M would find one more instance of a_0. (We start simulating M from above partial ID.) So we know k=i and k'=j. And Y is not random because we can reconstruct Y with less than $|Y\,|$ information. □ (of Lemma 2)

Remark 3: The conditions of Lemma 2 can be changed; for example, one head is in X of a_0, one head in X in *text*, and the third head is not in 1^k of a_0 and has a short $|k\,|/2$ description for its position. Since the idea of the proofs is the same, in the following if we meet similar situations we will simply refer to Lemma 2.

Lemma 3: For input $\#a_0\$text\cent$ where *text* is easy, if there is a time of M such that $p(h_3)>(1^k$ of $a_0)$, and $p(h_1)<\cent$, then Y is not random.

*Proof of Lemma*3: Append $a_1 0 a_2 0 a_3 a_4 0$ after *text* and before $\cent$, where $a_i=1^m X 1^m$ for $i=1,..,4$. Consider first time $p(h_1)=a_4 0$,

(1) h_1,h_2 did some matching, we are done by using the proof of Lemma 2; (Notice that h_3 is behind h_2.)

(2) x_4 of some $a_{i<5}$ is matched by (h_3,h_1) or (h_3,h_2), then $p(h_3)>(x_3$ of $a_0)$. Append $a_5 0 \ldots a_8 0$ after *text* and before $\cent$, and consider time $p(h_1)=a_8 0$,

(2.1) If there is an $i>4$ such that x_1 of a_i is matched to x_1 of some other a_j, then either $p(h_3)>\$$ or h_1,h_2 did the matching. In both cases, we are done by Lemma 2.

(2.2) If no x_1 of any $a_{i>4}$ is matched, we apply $\mathbf{P}(x_1)$:

(2.2.1) If S_1 is true, then $p(h_3)>\$$, we are done by Lemma 2.

(2.2.2) If S_3 is true, then trivially, x_1 in $a_{i>4}$ cannot be matched.

(2.2.3) If S_2 is true, $\mathbf{P}$ appends input with $a_f=1^{C_1+l*C_2}X1^m$, where $l=1$ at this moment and it is subjected to change. Continue to simulate M starting from $p(h_1)=1^{C_1+l*C_2}$, and consider the time when first x_1 is matched (for some $a_{i>5}$),

(2.2.3.1) The matching happened before h_2 reaches a_f: If the matching is done by h_3,h_1, then $p(h_3)>\$$, we can vary l and m in a_f to find k and k' as in Lemma 2, showing Y is not random; If it is done by h_1,h_2, then we can decide k and k' from $|k|$ information (for $p(h_3)$) plus $log\,|X|$ information by varying l and m in a_f as above; If the matching is done by h_3,h_2, then $p(h_3)>\$$, if when the matching happens $p(h_1)=\cent$, then we decide k from varying l, otherwise $p(h_1)<\cent$ and we apply lemma 2.

(2.2.3.2) h_2 reaches a_f before any matching of x_1 happens, but then 4 x_1's in a_5,a_6,a_7,a_8 are not matched to anything, trivial argument shows that some x_{1j} in some $a_{i>4}$ cannot be well-matched for this input.

(3) No x_4 of $a_{i<5}$ is matched, we apply $\mathbf{P}(x_4)$,

(3.1) If S_1 is true, apply cases (1) and (2);

(3.2) If S_3 is true, no more matching is possible;

(3.3) If S_2 is true, then, replacing x_1 by x_4, same argument of (2.2.3) can be applied except: in (2.2.3.1) if the matching is done by (h_3,h_1) then $p(h_3)>(x_3$ of $a_0)$, we apply the process of case (2); and if the matching is done by (h_3,h_2) then $p(h_3)>(x_3$ of $a_0)$, if $p(h_1)=\not c$ when the matching happens, we decide k from varying l, and if $p(h_1)<\not c$ at the time of matching, we again apply the process of (2). □ (of Lemma 3)

Lemma 4: In an input $\#a_0\$text\not c$, Let *text* be *easy*. If for some $j>0$, and $i=ab...c$ with each of $a,b,\cdots,c$ ranging from 1 to 6, $x_i<p(h_1),p(h_2)<\not c$, where x_i is in a_j, and this x_i is not well-matched (to a_0) and not matched to x_i of any a_l s.t. h_2 can still see x_i of a_l, and $|x_i|>|X|/1000$, then Y is not random.

Proof of Lemma 4: Consider first time $p(h_1)=text$ (just before $\not c$).

(1) If any matching of X happened by (h_3,h_2) or (h_3,h_1), then $p(h_3)>(1^k$ of $a_0)$ and we are done by Lemma 3.

(2) If h_3 did not join any matching of X, then x_i is not matched yet. If h_1 and h_2 are only separated by blocks of 1's, then x_i cannot be well-matched for this input.

(*) Now we apply $\mathbf{P}(x_i)$:

If S_1 is true, then $p(h_3)>(1^k$ of $a_0)$, we are done by Lemma 3;

If S_3 is true, then x_i cannot be well-matched for this input, contradicting to Lemma 1;

If S_2 is true, then $\mathbf{P}$ appended $a_f=1^{C_1+l*C_2}X1^m$ to the input. We continue to run M until $p(h_1)=(X$ of $a_f)$:

(2.1) If $p(h_3)$ involved in the matching of X of a_j, then we are done by Lemma 3. Otherwise,

(2.2) If some x_{ih} of a_f is not well-matched, then x_{ih} of a_j has to be at least matched once more by (h_2,h_3) before h_2 reaches a_f. (Otherwise, x_{ih1},x_{ih2},x_{ih3} of a_j and a_f cannot be all well-matched by only 2 heads left since x_i of a_j and a_f are not matched to each other.) But when the matching happens, if h_1 is at $\not c$, we can vary l to find k, by the method of Lemma 2, concluding Y is not random; If h_1 is not at $\not c$, we apply Lemma 3. (Or simply append $1^s X 1^t$ with smallest s and t to get k, k' .)

(2.3) If all x_{ij}'s of a_f are matched to some a_s's, then

(2.3.1) If h_3 joined matching, then $p(h_3)>(1^k$ of $a_0)$, we apply Lemma 3;

(2.3.2) If (h_1,h_2) matched all x_k's of a_f, then we take away the part $x_{i4,i5,i6}x_{i+1}\ldots x_6 1^m$ from a_f, then repeat $(*)$'s process in above recursively.

Notice that (2.3.2) above can not happen more than 10 times (no more than twice if one is careful) obviously since at least one a_i (with a full occurrence of X) has to be jumped over each time (2.3) is true. Also notice that in this process whenever h_3 joins the matching, we stop and apply Lemma 3. □ (of lemma 4)

Now we continue our proof of Theorem 1. Let the *text* be $a_1 0 a_2 0 \ldots a_6 0$ (not relevant to any a_i in above lemmas) temporarily, where $a_i = 1^m X 1^m$ for i=1,...,6. We will only consider the cases where $p(h_3)<(1^k$ of $a_0)$ because of Lemma 3. Consider the time $p(h_1)=a_1 0$.

(1) All x_i's of a_1 are matched (by h_1,h_2), then there is a time of M such that $p(h_1)=(x_2$ of $a_1)$, and $p(h_2)>(x_1$ of $a_0)$. Change a_1 to $1^m x_1 x_2$ and consider time $p(h_1)=a_2 0$ for the new input. There must exist an x_p in a_2 not matched to anything else yet (if $p(h_3)<(1^k$ of $a_0))$. Now consider $p(h_1)=a_6 0$:

(1.1) If, for $j=1,..,6$, all x_{pj} of a_2 are matched to some $a_{i>2}$'s, we find the smallest i such that x_{p4} of a_2 is matched to x_{p4} of a_i. Change *text* to

$$\ldots a_2 0 a_3 0 \ldots a_{i-1} 0 1^m x_{p1} \ldots x_{p4} 0 \not c$$

and simulate M until $p(h_1)=text$.

Claim 1.1: Let a_s be the first such that x_{p1} of a_2 matches x_{p1} of a_s, then for $j=2,..,6$, x_{pj} in any $a_{t \geq s}$ is not well-matched yet.

We then apply $\mathbf{P}(x_p)$: If S_1 is true, then $p(h_3)>(1^k$ of $a_0)$, we apply lemma 3; If S_3 is true, x_{p5} of a_2 cannot be well-matched, contradicting to Lemma 1; If S_2 is true, $\mathbf{P}$ adds $a_f = 1^{C_1 + l * C_2} X 1^m$ to the input. Consider time $p(h_1)=$(X of a_f). ($p(h_3)$ should be still $<(1^k$of $a_0)$, otherwise we are done by lemma 3.)

(1.1.1) (h_1,h_2) did not match $x_{p5}x_{p6}$ of a_f to that of a_2, and $p(h_2)>(x_{p5}$ of $a_2)$, then x_{p5} and x_{p6} of a_2 have not been matched to anything yet, we apply Lemma 4;

(1.1.2) (h_1,h_2) matched one of x_{p5},x_{p6} of a_2 to a_f, or $p(h_2)<(x_{p5}$ of $a_2)$, then $x_{p1}x_{p2}x_{p3}$ of a_2,a_f are not matched to each other ($x_{p1}x_{p2}x_{p3}$ of a_f is not matched to anything at all.) because when $p(h_1)=(x_4$ of $a_i)$ we have $p(h_2)>(x_3$ of $a_2)$. Notice that at this moment, for some $j<4$ x_{pj} of a_2 is not well-matched yet, and neither are x_{p1},x_{p2},x_{p3} of a_f. They should be well-matched by (h_2,h_3). But some of x_{pj1},x_{pj2}, or x_{pj3} in a_2 has to be matched (to something) by (h_2,h_3) before h_2 goes to a_f since if h_2 reaches a_f before doing any matching and then to make x_{pj1},x_{pj2}, and x_{pj3} of a_f well-matched one of x_{pj1},x_{pj2}, or x_{pj3} of a_2 cannot be well-matched at the same time by Claim 1.1 and a straightforward exclusion argument. So we assume (h_2,h_3) do some matching before h_2 goes to a_f. But by the time of this matching, if h_1 is not at $\cent$, we are done by Lemma 3; and if $p(h_1)=\cent$ we can vary l to find k by the method of Lemma 2, showing Y is not random.

(1.2) If there exists j such that x_{pj} is not matched to any $a_{i>2}$, then,

(1.2.1) if there are $h,q>2$ such that x_{pj1} and x_{pj2} in a_h and a_q are matched, then $p(h_2)>a_2$, we apply Lemma 4;

(1.2.2) assuming $p(h_2)\le a_2$, that is, above (1.2.1)'s condition is not true, we apply $\mathbf{P}(x_p)$:

(1.2.2.1) If S_1 is true, apply Lemma 3;

(1.2.2.2) If S_3 is true, no more matching possible, contradicting to Lemma 1;

(1.2.2.3) If S_2 is true, we consider time $p(h_1)=(X$ of $a_f)$ (appended by $\mathbf{P}$), if h_3 is involved in the matching before h_2 reaches a_f we are done by Lemma 3; Otherwise, by the condition of (1.2.2) we can derive that there exists k=1 or 2, s.t. for at least 2 $a_{i>2}$'s, two x_{pjk}'s are not well-matched and they are not matched to each other. We go on simulating M. If h_2 does not do any matching with h_3 before going to a_f, then x_{pjk1} or x_{pjk2} in one of the a_i's can not be well-matched (by a trivial argument). If (h_2,h_3) do some matching before h_2 reaches a_f, then as before, at the moment of the matching, if $p(h_1)=\cent$, we vary l to find k, concluding that Y is not random (by the method of lemma 2); if $p(h_1)<\cent$, we apply Lemma 3.

(2) Some x_p in a_1 is not matched. we consider time $p(h_1)=a_6 0$, exactly same argument as in (1.1)&(1.2) applies. (Change a_2 to a_1.) □ (of Theorem 1)

Remark: The result here does not imply the result of [LY] directly since in above proof we had to consider many occurrences of pattern in the text. We hope the idea of easier text and harder pattern can suggest some possible approach to the general $k>3$ case.

4. String-matching by a 2-way k-DFA with k-1 heads blind

A 2-way k-DFA is just like a k-DFA but each head can go both directions. We assume that a 2-way k-DFA stops by entering a final state. A head is *blind* if it can see only end-markers.

In [DG] it is proved that 2-way 2-DFA with one head blind cannot do string-matching. Obviously 2-way 3-DFA with 2 heads blind can do string-matching. Here in contrast to the impossibility result of Theorem 1, we prove a lower bound on the time to do string-matching required by a 2-way k-DFA with k-1 blind heads. And we will also give an upper bound for some simple matching problem. We hope this can shed some light on the other important open problem concerning the lower bound of doing string-matching by a 2-way 2-DFA.

Theorem 2: String-matching requires $\Omega(n^2/logn)$ time for a k-head two way DFA with k-1 heads blind, where n is the length of the input.

Proof: Suppose M does string-matching in $o(n^2/logn)$ time, where M is a 2-way k-DFA with k-1 blind heads. Let h_1 be the non-blind head and $h_2, \ldots, h_k$ be the k-1 blind heads. Fix a long enough K-random string X and consider input

$$\#X\$0^{|X|}X¢$$

on the input tape. Define the crossing sequence (c.s.) of h_1 under a fixed position of the input tape of M to be a sequence of items of form (state of M, positions of $h_2, \ldots, h_k$) which specifies the status of M when h_1 passes this position. Now consider $|X|$ c.s.'s under $0^{|X|}$. If each one is of length greater than $|X|/k^2log|X|$, then M takes $O(n^2/logn)$ time, a contradiction; Otherwise there exists a c.s. of length less than $|X|/k^2log|X|$. But than from this c.s., which can be described by less than $|X|/2$ information, we can reconstruct X by a short program as follows: For each X', form input $\#blank\$0^{|X|}X'¢$. Start to simulate M from the first item of our short c.s., going only to the right. Each time h_1 runs back to the position of this c.s. we match the current status of M against the next item of the c.s. and if they match we take the following item in the c.s. and continue to simulate M from it (Right turn only!). Thus if the simulation finishes with everything matches we can conclude X'=X. □ (of Theorem 2)

One may wonder whether this *logn* factor can be canceled. For this input the answer is NO, as the next theorem demonstrates.

Theorem 3: 2-way 3-DFA with 2 heads blind can accept L={#x$yx¢} in time $O(n^2/logn)$.

Remark: It is proved in [LY] that L defined above cannot be accepted by a 1-way 2-DFA. By a similar proof the language L'=$\{\#a_0\$a_1*a_2*...*a_l¢ \mid a_0=a_i$ for some $i\}$, defined and shown to be not acceptable by a 2-way 2-DFA with one head blind in [DG], is acceptable in time $n^2/logn$ by a 2-way 4-DFA with three blind heads.

Proof : We sketch an M accepting L with one regular head h_1 and two blind heads h_2 and h_3. To match fast, M will match block by block with each block roughly *logn* long, where n is the length of input. Algorithm for M follows,

Step_0: h_1,h_2,h_3 at # sign. h_2 (the counter) moves one step right.

Step_i:

Step_i. 1: h_1 moves right one step, and (h_2,h_3) double (times 2) the size of the counter held by h_2 (still held by h_2 after modification). h_2 moves right one more step (plus 1) if h_1 reads a one; If h_1 reads a zero h_2 does not move. If h_2 did not pass the ¢ sign, then repeat *step_i.* 1, else go to *step_i.* 2. {Remember a *logn* long block in a counter held by h_2.}

Step_i. 2: h_2 holds the counter and h_3 goes to the $ sign. h_1 and h_3 move simultaneously to the left until h_1 reads # sign. h_1 and h_3 switch positions. Then h_1 goes to ¢ sign. h_1 and h_3 go backward at the same time until h_3 reads # sign. {This places h_1 in the corresponding matching position in the *text*.}

Step_i. 3: h_2 and h_3 decrease the counter, held by h_2, by half (divide by 2). If there is a remainder 1 then h_1 mush read a 1; If the division is even, then h_1 must read a 0. h_1 moves one step left. Repeat *Step_i.* 3 until the h_2 counter becomes zero. {matching a block}

Step_i. 4: h_1 goes back to the corresponding position for the next block in the pattern with the help of h_2 and h_3. To do this, remember current (after *step_i.* 3) distance between h_1 and the ¢ sign. Send h_1 back to *pattern* with the same distance to the \$ sign. This is actually the position of h_1 before *Step_i* is started. Then repeat the process of *Step_i.* 1 once more.) Go to *Step_i*+ 1. {Re-set h_1 for next step ($i+1$).}

end.

By standard calculation, *Step_i* is repeated no more than $|pattern|/logn$ times. For each repetition, *Step_i* requires O(n) time. Therefore total is O(n^2). The correctness of this algorithm is trivial to see. □

5. Probabilistic checking is easier than probabilistic generating

It has been an interesting philosophical question [W]: Is (probabilistic) checking easier than (probabilistic) generating? For example, given matrix A, B, and C, Freivalds showed (see [W]) that we can probabilistically check AB=C in n^2 time, but no one knows how to calculate AB better than the Strassen's or Pan's algorithm probabilistically (open problem 2.6 in [W]). Also similarly it is known [W] that given polynomials $p_1(x), p_2(x), p_3(x)$, the probabilistic checking of $p_1(x)p_2(x)=p_3(x)$ can also be done faster than the known generating ($p_3(x)$) algorithms. Here we shall provide an example which does show that checking is easier than generating.

We will prove a lower bound which says a block cannot be moved faster than n^2 time even with the help of a random number generator. We follow [G], a PTM is a TM equipped with a random number generator. A PTM decides the next move by a random choice from two possible branches. Language L is accepted by a PTM P in time $t(n)$ if there exist an $\epsilon<1/2$ such that if $x \in L$ then P accepts x in with probability greater than 1-ϵ in time $t(n)$, otherwise P accepts x with probability less than ϵ in time $t(n)$. In this section, we will solely consider the 1-tape probabilistic machines (1-tape PTM's) without an extra input tape, i.e., the input is presented on this single work tape at the beginning of the computation.

It is a very interesting result by Freivalds [F] that a one tape PTM can match two strings on 1 tape in time $O(nlogn)$. In contrast we show the following.

Theorem 4: Consider a 1 tape PTM M, with input $x\#^{|x|}0^{|x|}$ presented on its only working tape. We want M to move x to the 0's positions, i.e., output $x\#^{|x|}x$ where $x\#^{|x|}$ stays at original position. Then for any fixed $\epsilon<1/2$, ~~to do~~ this with error probability ϵ, M requires $\Omega(n^2)$ time.

Proof of Theorem 4: Assume M does the job in $o(n^2)$ time. We fix a K-random string x of enough length. Consider the crossing sequences (c.s.) at the # signs. If the number of computations is T (for all random sequences), then each computation uses $o(n^2)$ time, in total it causes $o(n^2)*T$ elements of the c.s. at # signs. Let $\epsilon=1/2-\delta$. Then there must exists a position i at some # sign, s.t. the c.s.'s at i are short, $o(n)$ long (say, $n/(10|M|)$), for T'=(1-δ)T computations, since otherwise the total # of c.s.'s for all computations is going to be $O(n^2T)$, a contradiction.

Suppose, we have above i. Then for those T' computations, the c.s.'s at position i are shorter than $n/10|M|$. Notice that each c.s. corresponds to one or more computations that causing this c.s.. Among these T' c.s.'s (there might be a lot same c.s.'s), at least one c.s. corresponds to those computations where more than half of them produce correct output, since otherwise the error probability exceeds $(1-\delta)/2>\epsilon$.

But if above c.s. exists, we can give a short program to produce x as follows, we generate all possible random strings to run M, pick out those computations which match the c.s. at position i. The majority of these computations should output x at the $0^{|x|}$ position, and our short program will output this x. The information we need for this program is the c.s. of length n/10, logn for the #'s and 0's, total less than n, contradicting to the K-randomness of x. □ (of Theorem 4)

Remark: Comparing to the nlogn probabilistic algorithm for accepting $x\#^{|x|}x$ (with any fixed small error ϵ) on 1 tape by Freivalds [F], this lower bound gives us an interesting conclusion: checking is indeed easier than generating. Notice that this is not true for 1-tape deterministic or nondeterministic machines since a n^2 lower bound for accepting the palindromes were proved long time ago by Hennie.

6. Open questions

Clearly the important open problems are:

(1) Prove, for all k, k-DFA cannot do string-matching; and

(2) Give nontrivial lower or upper bounds for string-matching by a 2-way 2(3,4,5,)-DFA.

7. Acknowledgements

The author would like to thank Professors Zvi Galil, Juris Hartmanis, Joel Seiferas, and Yaacov Yesha for encouragement and helpful discussions.

8. References

[BM] R.S. Boyer and J.S. Moore, A fast string searching algorithm, CACM 20, 10 (Oct. 1977) pp. 762-772.

[DG] P. Duris and Z. Galil, Fooling a two-way automaton or one pushdown store is better than one counter for two way machines, Proceedings 13th ACM STOC (1981) pp. 177-188.

[F] R. Freivalds, Probabilistic machines can use less running time, Information Processing, 77.

[G] J. Gill, Computational complexity of probabilistic Turing machines, SIAM J. Comp. 6 (1977) pp.675-695.

[GS] Z. Galil and J. Seiferas, Time-space optimal string-matching, Proceedings 13th ACM STOC (1981) pp.106-113.

[HU] J. Hopcroft and J. Ullman, Introduction to automata theory, languages, and computation, Addison-Wesley (1979).

[KMP] D.E. Knuth, J.H. Morris, Jr., and V.R. Pratt, Fast pattern matching in strings, SIAM J. Comp. 6, 2 (Jun. 1977) pp. 323-350.

[LY] M. Li and Y. Yesha, String-matching cannot be done by a two-head one-way deterministic finite automaton, TR 83-579, Department of Computer Science, Cornell University. (1983)

[P] W. Paul, Kolmogorov complexity and lower bounds, 2nd International conference on fundamentals of computation theory. (1979)

[W] D.J.A. Welsh, Randomized Algorithms, Discrete Applied Math. 5 (1983) pp 133-145.

[YR] A. Yao and R. Rivest, k+1 heads are better than k, J. ACM, 25 (1978), pp. 337-340.