数理解析研究所講究録
454巻 1982年 23-44

A NEW AUTOMATON MODEL SUITABLE FOR MAXIMAL COMMON SUBSTRING COMPUTATION AND ITS APPLICATION TO DATA COMPRESSION

Narao NAKATSU*, Yahiko KAMBAYASHI** and Shuzo YAJIMA**

* Center for Educational Technology, Aichi University of Education

** Faculty of Engineering, Kyoto University

1. Introduction

In large database systems, we usually encounter the situation when a set of similar data is to be stored and only one of these data, a reference datum, is referred often(versions of programs; the most recent version is referred often, monthly data of some organizations; the most recent datum is referred often). In such cases, data compression procedures utilizing the similarity of data seem to be promising.

Rodeh et al.[11] have proved that a data compression procedure utilizing repeated substrings can give optimal encoding scheme as the length of the input string grows to infinity. The authors have presented efficient data compression procedures for a set of similar data[5]. Our compression methods are one of the differential file approach, where data are expressed by the differences from a reference datum when they are similar.

A key factor of efficient realization of such procedures is the computation of maximal repeated substrings(P1) or maximal common substrings(P2). These procedures are well known pattern matching problems. A native algorithm for the solution of problem P1 and P2 takes $O(n^2)$ and $O(mn)$

time respectively, where m and n are lengths of given strings.

The position tree[1](which is called a prefix tree or suffix tree by some researchers) is the previously known best result for the above problems. Once a compacted version of the tree is constructed in linear time, P1 and P2 can be solved in linear time by traversing the tree. Weiner showed an algorithm for constructing a compacted position tree in linear time[12]. More simpler and more space-economical procedure was introduced by McCreight [10].

These algorithms are <u>off-line</u> algorithms and Majster et al.[9] have presented an <u>on-line</u> construction algorithm but it takes $O(n^2)$ time in the worst case. Moreover, the position tree itself is not so efficient in the case when all maximal common substring between a reference datum, s_0, and other data, $s_i (1 \leq i \leq N)$, should be calculated. In this case, position tree should be constructed for each string $s_0 s_i$ $(1 \leq i \leq N)$ or a position tree for the string $s_0 s_1 \ldots s_N$ should be constructed, which is time-consuming or space-consuming, respectively.

In this paper, a new automaton model called an auxiliary-memory automaton(AMA for short) is introduced. An SMM, one of AMA, which accepts all substrings of a given string w of length n can be constructed in the time proportional to n, while the construction of a conventional finite automaton requires the time proportional to n^2. This approach is better than the position tree in the following points.

(1) The construction algorithm is an on-line algorithm and a linear time algorithm.

(2) To find all maximal common substrings for two strings, the procedure proposed in this paper requires less storage space.

(3) For efficient data compression, we need to calculate all maximal common

substrings between each string and the reference string. The proposed procedure is efficient since we only need to construct one SMM to accept all substrings of the reference string and it is repeatedly used to calculate maximal common substrings for other strings.

2. Basic concepts

In this paper, following notations are used. An alphabet Σ is a finite set of symbols. A string over Σ is a finite-length sequence of symbols from Σ. A concatenation of strings x and y is the string xy. The length of a string w, denoted by $|w|$, is the number of symbols in w. A string w may be represented by $w(1)w(2)\ldots w(n)$ $(w(i)\in\Sigma,\ 1\leqq i\leqq n=|w|)$ and a substring $w(i)w(i+1)\ldots w(j)$ is denoted by $w(i:j)$. A position in a string w is an integer between 1 and $|w|$. The symbol $a\in\Sigma$ occurs in position i of string w if $w=yaz$ with $|y|=i-1$. Empty string is denoted by ε. Σ^+ is a set of strings over Σ except ε. For a string w of length n, $w(k)$ $(k<1$ or $k>n)$ is considered as ε unless noted.

[Definition 1] Consider two strings w_1 and w_2. $w_1(0)$ and $w_1(|w_1|+1)$ are considered as a special symbol $\#(\notin\Sigma)$ in this definition. A substring u of w_1 is called a repeated substring of w_1 iff u appears twice at least in w_1. A string u is called a common substring of w_1 and w_2 iff u is a substring of both w_1 and w_2. A repeated substring $w_1(i:j)$ is called a maximal repeated substring (MRS for short) when neither $w_1(i-1:j)$ nor $w_1(i:j+1)$ is a repeated substring of w_1. A common substring $w_1(i:j)$ is called a maximal common substring (MCS for short) of w_1 w.r.t. (with respect to) w_2 iff neither $w_1(i-1:j)$ nor $w_1(i:j+1)$ is a common substring of w_1 and w_2.

[Example 1] Consider the following two strings.

$$\begin{array}{rcccccccc} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ w_1= & b & a & a & a & b & a & a & a \end{array} \qquad \begin{array}{rccccccccc} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ w_2= & a & b & b & b & a & b & a & a & a \end{array}$$

$w_1(1:4)$ is an MRS of w_1 and it is also an MCS of w_1 w.r.t. w_2. $w_2(6:9)$(= $w_1(1:4)$) is not an MCS because $w_2(5:9)$ is an MCS of w_2 w.r.t. w_1.

[Definition 2] Let $w=w(1)w(2)...w(n)$ be a string over an alphabet Σ. Let $w'=w\$$ ($\$$ is an endmarker of w and $\$$ is not in Σ). A position identifier W_i for position i in w' is the shortest substring u of w' such that

(i) $w'=yuz$ and $|y|=i-1$,

(ii) u is not a repeated substring of w'.

If W_i is given, we can uniquely identify the starting position of W_i in w'.

[Definition 3] A compact position tree T for a string $w=w(1)w(2)...w(n)$ is a tree which satisfies following conditions. Let $w'=w\$$.

(i)T has n+1 leaves labeled 1,...,n+1. The leaves of T are one-to-one correspondence with the positions in w'. Each edge of T is lebeled by a string over $\Sigma \cup \{\$\}$.

(ii)Every interior node of T has two sons at least.

(iii)For each node N in T, the edges leaving N have labels whose first symbols are distinct from one another.

(iv)The string obtained by concatenating labels on the path from the root to the leaf i equals the position identifier W_i.

Note that there is exactly one compact position tree for each string.

[Example 2] The compact position tree for the string babaa is given in Fig.1.

Weiner showed an <u>off-line</u> construction algorithm for a compact position tree, which can be utilized for the calculation of MRSs and MCSs.

[Proposition 1] For given two strings w_1 and w_2, all MCSs between w_1 and w_2 can be obtained in time of $O(|w_1|+|w_2|)$ by the compact position tree for

the string $t=w_1\lambda w_2$ $(\lambda \notin \Sigma)$.

Weiner's algorithm is off-line and a known on-line construction algorithm requires time of $O(|w|^2)$[9]. In the following section, an online construction procedure of an SMM which is functionally equivalent to the position tree is presented.

3. A new automaton model, auxiliary-memory machine

3.1 Auxiliary-memory automaton

In this section, a new automaton model, an auxiliary-memory automaton, is defined. A machine, SMM (Substring Matching Machine), one of auxiliary-memory machine, is introduced for computing MRSs and MCSs efficiently. This machine is functionally equivalent to the position tree. An online and linear construction algorithm of an SMM for a given string is shown in this section. Previous algorithms are offline or require time proportional to the square of the length of a given string. By the SMM, an online linear-time data compression procedure can be obtained.

[Definition 4] An auxiliary-memory automaton (AMA) is defined by 8-tuple $(S,\Sigma,\delta,\mu,A,I,s_0,c_0)$, where S is a finite set of states, Σ is a finite input alphabet, s_0 in S is the initial state, A is an auxiliary-memory and contains one element of I which is a set of nonnegative integers, c_0 is an initial value of A, δ(the next state function) is a mapping $S\times\Sigma\times I \rightarrow S$ and μ is a mapping $S\times\Sigma\times I \rightarrow I$. The configuration of an AMA is described by (s,c), where $s\in S$ is a state and $c\in I$ is a content of A.

Online algorithm is obtained by using a backward position identifier defined as follows.

[Definition 5] A backward position identifier (BWPI for short) for position i in a string w is a substring $w(j:i)$ $(j\leq i)$ which is not a repeated substring

of w(1:i). The minimum length BWPI for position i is denoted by MBW_i or MBW s_i.

[Definition 6] An AMA with n+1 states $s_0, s_1, \ldots, s_n$ is a substring matching machine (SMM for short) for w(|w|=n) iff it satisfies the following condition.

Condition: When a sequence whose longest suffix is a $BWPI_i$ for some i ($1 \leq i \leq n$) is applied, the machine enters s_i and A has $|BWPI_i|$. (It is assumed that the machine starts from s_0 with 0 in A). In this case, $BWPI_i$ is said to be accepted at s_i. When a sequence whose suffixes are not $BWPI_j$ for any j is applied, the machine enter s_0 and A has 0.

The following proposition holds.

[Proposition 2] M is an SMM for a string w iff M has n+1 states $s_0, s_1, \ldots, s_n$, and δ and μ are defined as follows.

(a) If x appears in w, $(\delta(s_i,x,c), \mu(s_i,x,c))=(s_j,k)$, where k is the maximal number such that $w(i-k+2:i)x=w(j-k+1:j)$ ($1 \leq k \leq c+1$, $0 \leq i,j \leq n$) and j is the minimum number that satisfies this condition.

If x does not appear in w, $(\delta(s_i,x,c),\mu(s_i,x,c))=(s_0,0)$.

(Proof) --> Consider an SMM M. If x does not appear in w, a string which has x as its last symbol cannot be $BWPI_i$ for any i. So the machine enters s_0 and A has 0 from the definition of the SMM. Suppose that M is in s_i and A has c ($i \geq 0$). Because M is an SMM, a $BWPI_i$ of length c has just been applied to M. If $BWPI_i x=BWPI_j$ then M must enter s_j and A has c+1. So $\delta(s_i,x,c)=s_j$ and $\mu(s_i,x,c)=c+1$ hold. Otherwise, consider the longest suffix, y, of $BWPI_i x$ s.t. y is $BMPI_j$ for some j. It must be accepted in s_j and A contains $|BWPI_j|$. So $\delta(s_i,x,c)=s_j$ and $\mu(s_i,x,c)=|BWPI_j|$. The case that w(i-c+1:i) is not $BWPI_i$ cannot happen from the definition of SMM. We can define δ and μ arbitrarily in such a case. Then it is proved that an SMM

satisfies the condition (a).

<-- Consider a machine, M, that satisfies the above condition(a). Consider a string x of length 1 ($x \in \Sigma^+$). If $\delta(s_0,x,0)=s_j$ and $\mu(s_0,x,0)=1$ then x is BWPI $_j$. Assume that the machine M satisfies the definition 6 for the input w of length i. Assume that M is in s_j and A has c. By the assumption, w(j-c+1:j) is a $BWPI_j$. If $\delta(s_j,x,c)=s_h$ and $\mu(s_j,x,c)=c+1$ then it is easily observed that w(j-c+1:j)x is $BWPI_h$. If $\delta(s_j,x,c)=s_h$ and $\mu(s_j,x,c)=k(\leq c)$ then it is observed that w(j-k+2:j)x is a $BWPI_h$ of length k. If $\delta(s_i,x,c)=s_0$ then x does not appear in w. So M is an SMM for the input of length i+1. (Q.E.D.)

To implement δ and μ, the concept of a labeled transition diagram and two functions are introduced.

[Definition 7] A labeled transition diagram for a string of length n is a labeled directed graph of n+1 nodes. Each node corresponds to each state in $\{s_0, s_1, \ldots, s_n\}$. Each directed edge is labeled by a symbol $x \in \Sigma$ and an interval of I which is a subinterval of $[0,\infty]$.

Interpretation of a labeled transition diagram is as follows. If there exists a directed edge from s_i to s_j labeled by x and $[i_1,i_2]$, the transition from s_i is effective when the current state is s_i, the input symbol is x and the content of A is in $[i_1,i_2]$. The transition function δ' is defined to represent the next state on the labeled transition diagram. If there is a directed edge from s_i to s_j labeled by x and $[i_1,i_2]$, then $\delta'(s_i,x,c)=s_j$ for all c in $[i_1,i_2]$ and $\delta'(s_i,x,c)=\hat{}$(undefined) if there is not such an edge.

[Definition 8] Consider the string w=w(1)w(2)...w(n) and the SMM of w. For each state s_i ($1 \leq i \leq n$), a reset function r and a position function p are defined as follows.

$r(s_i)=|MBW_i|-1.$

$p(s_i) = s_j$, where j is the minimum number such that $w(i-r(s_i)+1:i)=w(j-r(s_i)+1:j)$ and $1 \leqq j \leqq i$ when $r(s_i) \neq 0$

$= s_0$ when $r(s_i)=0$.

$p^{(m)}(s_i)$ and $r^{(m)}(s_i)$ are defined as follows: (i) $p^{(1)}=p$ and $r^{(1)}=r$, (ii) $p^{(m)}(s_i)=p(p^{(m-1)}(s_i))$ and $r^{(m)}(s_i)=r(p^{(m-1)}(s_i))$ for $m>1$. $p(s_0)$ and $r(s_0)$ are undefined(^). So $p(\hat{})$ and $r(\hat{})$ are also undefined.

[Example 3] Consider the string w=b b a a b b a b a. The reset function and position function of the SMM of w are shown in Fig. 2. For example, $r(s_3)=0$ and $p(s_3)=s_0$ because $w(3)=a$ is an MBW_3. $r(s_6)=2$ and $p(s_6)=s_2$ because $w(4:6)=abb$ is an MBW_6.

The functions δ and μ of an SMM are expressed by δ'(The transition function on the labeled transition diagram), r and p as follows. Suppose that the present sate is s_i and an input symbol is x and the content of A is c.

(1) If $\delta'(s_i,x,c)$ is defined (i.e. corresponding edge exists), then the SMM follows this transition and the content of A is incremented by one.

(2) Find the minimum m such that $\delta'(p^{(m)}(s_i),x,r^{(m)}(s_i))$ is defined. If such m exists, the next state is $\delta'(p^{(m)}(s_i),x,r^{(m)}(s_i))$ and A is set to be $r^{(m)}(s_i)+1$. Otherwise, the machine enters s_0 and A has 0.

The following example shows an example of an SMM defined by δ',r and p. An operation of the SMM is also presented.

[Example 4] The labeled transition diagram of the SMM of the string w=b b a a b b a b a is shown in Fig. 3. Whenever an input symbol is applied to the SMM, the machine makes a transition. If the next state is defined on the labeled transition diagram, the machine enters a new state according to the labeled transition diagram and the content of A is incremented by 1. If there is no transition egde corresponding to the input x at state s_i with c in A, then the machine enters the state $p(s_i)$ and the content of A is set to be $r(s_i)$. Then the machine continues to make transition by the input x recursively

until next state is defined or the machine visits s_0 twice. For example, suppose that the machine is in s_3. When "a" is applied, the machine enters s_4. When "b" is applied, the machine enters s_5 if the content of A is 1 and s_8 if A contains 2 or 3, respectively. In other cases, next state is undefined on the labeled transition diagram and the machine enters $f(s_3)=s_0$ and the content of A is set to be $r(s_3)=0$. Continuing the transition at s_0 by the same input symbol, the machine finally enters s_1(for the input "b") or s_3 (for the input "a") or s_0 (for other symbols).

The formal construction algorithm of the SMM for a string w is shown in Procedure 1.

[Procedure 1] Construction of an SMM

(1) For a given string w=w(1)w(2)...w(n), make s_0 and s_1 and make a transition from s_0 to s_1 labeled by w(1) and an interval [0,∞].

(2) $p(s_1)$<--s_0, $r(s_1)$<--0.

(3) For i=2 to n do begin

make a state s_i and make a transition form s_{i-1} to s_i by w(i) and [0,∞]. PSTATE<--s_{i-1}, ASTATE<--p(PSATAE), call CONST(ASTATE,PSTATE,i)

end

Procedure CONST(ASTATE,PSTATE,i)

if δ'(ASTATE,w(i),r(PSTATE)) is defined

then begin $p(s_i)$<-- δ'(ASTATE,w(i),r(PSTATE))

$r(s_i)$<-- r(PSTATE)+1. end

else begin if δ'(ASTATE,w(i),j) is undefined for any j>0

then begin make a transition from ASTATE to s_i by w(i).

if ASTATE=s_0 then begin $p(s_i)$<--s_0, $r(s_i)$<--0. An interval [0,∞] is labeled to the new edge. end

else begin An interval [r(ASTATE)+1,r(PSTATE)] is lebeled to the new edge. call

CONST(p(ASTATE),ASTATE,i). end

end

else begin find the maximum j such that δ'(ASTATE,w(i),j) is defined. For such j, $p(s_i)$<--δ'(ASTATE,w(i),j), $r(s_i)$<--j+1. make a transition from ASTATE to s_i by w(i) and the interval [j+1,r(PSTATE)] is labeled. end

end <u>end of procedure CONST</u>

We show that the machine constructed by Procedure 1 is an SMM.

For Procedure 1, next two theorems hold.

[Theorem 1] The machine constructed by Procedure 1 is the SMM for w under the interpretation of δ',p and r.

(Proof) Consider the machine for w=w(1) constructed by Procedure 1. This machine is shown in Fig. 4. It is trivial that this machine is an SMM for w(1). Assume that the SMM for w (|w|=i) is correctly realized by the machine constructed by Procedure 1. Let us consider the machine for wa. There is an edge created from s_i to s_{i+1} labeled by "a" and $[0,\infty]$. Suppose that w(i-h+1:i)=w(j-h+1:j) and $\delta'(s_j,a,t)$ is defined for some t, where h is the maximal number that satisfies this condition and j is such a minimum number. If such h does not exist, let h and j be 0. From the definition of r and p, for some m, $p^{(m)}(s_i)=s_j$ and $r^{(m)}(s_i)=h$ hold. From the definition, δ and μ of the SMM for w should be changed at the states $p^{(1)}(s_i),\ldots,p^{(m-1)}(s_i)$, i.e. $(p^{(k)}(s_i),a,c)=s_{i+1}$, $|MBW_{p^{(k)}(s_i)}|\leq c\leq|MBW_{p^{(k-1)}(s_i)}|-1$ $(1\leq k\leq m-1)$. (Note that $MBW_{p^{(k)}(s_i)}=r^{(k+1)}(s_i)+1$). This is realized by the edge from $p^{(k)}(s_i)$ to s_{i+1} labeled by "a" and [r(ASTATE)+1,r(PSTATE)]. By these edges, $BWPI_{i+1}$s whose lengths are not less than $|MBW_{p^{(m-1)}(s_i)}|+1$ are accepted at s_{i+1}. To implement δ and μ at s_{i+1}, the maximal value v s.t. w(i+1-v+2:i+1)x=w(u-v+1:u) should be found. I show this is realized by δ', p and r. Consider Fig. 5. There are 3 cases to be considered.

(1) $w(j+1)=a$: In this case, we need not alter δ and μ at s_j. To realize δ and μ at s_{i+1}, it is easily proved that $p(s_{i+1})=s_{j+1}$ and $r(s_{j+1})=r^{(m)}(s_i)+1=|MBW_{p^{(m-1)}(s_i)}|$. Then all $BWPI_{i+1}$s are accepted at s_{i+1}.

(2) $\delta'(p(s_j),a,r^{(m)}(s_i))$ $(=s_h)$ is defined: δ and μ need not be altered at s_j. $p(s_{i+1})=\delta'(s_j,a,r^{(m)}(s_i))$ and $r(s_{i+1})=r^{(m)}(s_i)+1$ hold because $w(h-r^{(m)}(s_i):h)=w(i+1-r^{(m)}(s_i):i+1)$ hold from the assumption.

(3) $\delta'(s_j,a,c)$ is defined for $c<p^{(m)}(s_i)$: Let c be such maximal number and $\delta'(s_j,a,c)=s_h$. If $\delta'(s_{j'},a,c')$ is defined for some $c'\geqq c$ and $j'<j$, the machine for w becomes not to be an SMM. So $p(s_{i+1})=s_h$ and $r(s_{i+1})=c+1$ hold to realize δ and μ at s_{i+1}. To accept all $BWPI_{i+1}$ at s_{i+1}, an edge from s_j to s_{i+1} labeled by "a" and $[c+1,r^{(m)}(s_i)]$ is created in Procedure 1. (Q.E.D.).

[Theorem 2] The SMM of w can be constructed by Procedure 1 in time of $O(|w|)$.

(Proof) In Procedure 1, subprocedure CONST is called n-1 times and n+1 edges are constructed in main procedure. When the procedure CONST is called, there are 3 cases to be considered.

1) Transition δ' is made.

2) A new edge is created and CONST is called.

3) A new edge is created and a transition δ' is made.

1) and 3) are executed in constant time. We show that CONST is called at most $2|w|$ times in Procedure 1. Consider $r(s_i)$ and $r(s_{i+1})$. If $r(s_{i+1})=r(s_i)+1$ then CONST is called only once. If $r(s_i)\geqq r(s_{i+1})$ then CONST is called at most $|r(s_i)-r(s_{i+1})|+1$. By this fact, CONST is called at most $n-1+(n-r(s_n))$ times. We can also prove the fact that the number of edges in the labeled transition diagram does not exceed $2n-r(s_n)$. Creation of a new edge and a transition can be executed in constant time. By this fact, Procedure 1 requires time of $O(|w|)$. (Q.E.D.)

We can find all MRSs and MCSs efficiently. MRSs of w are expressed by the functions p and r of the SMM for w. To calculate MCSs of w_1 w.r.t. w_2, we only need to apply w_2 to the SMM of w_1. From the definition of the SMM, if the SMM of w_1 has a configuration (s_j,c) after applying $w_2(1:i)$, then $w_1(j-c+1:j)=w_2(i-c+1:i)$ holds. By this observation, all MCSs of w_2 w.r.t. w_1 can be found efficiently by an SMM. A formal procedure for this purpose is presented in the follwing section.

The next example shows how Procedure 1 works.

[Example 5] The SMM for w=bba is shown in Fig. 6-(a). Consider the SMM for wa. An edge labeled by "a" and $[0,\infty]$ from s_3 to s_4 is created. $\delta'(p(s_3),a,r(s_3)]=\delta'(s_0,a,0)=s_3$. So $p(s_4)=s_3$ and $r(s_4)=r(s_3)+1=1$. Then consider the SMM for wab. An edge labeled by "b" and $[0,\infty]$ from s_4 to s_5 is created. $\delta'(p(s_4),b,r(s_4))=\delta'(s_3,b,1)$ is undefined. Then we must create a new edge from s_3 to s_5, which is labeled by b and $[r(s_3)+1,r(s_4)](=[MBW_3,MBW_4-1]=[1,1])$. Then $\delta'(p^{(2)}(s_4),b,r^{(2)}(s_4))=\delta'(s_0,b,0)$ is tested. $\delta'(s_0,b,0)=s_1$. So $p(s_5)=s_1$ and $r(s_5)=p^{(2)}(s_4)+1=1$. The SMM for wab is shown in Fig. 6-(b).

3.2 Required space

McCreight has presented a position tree which requires less storage space than the one presented by Weiner. For the storage space, next proposition holds[10].

[Proposition 3] The Suffix tree(called by McCreight) for the string w requires $4n \log n+3n \log|\Sigma|+4n$ bits[10].

McCreight uses hash function to express the connection between nodes of the tree. The SMM constructed in 3.1 can be realized by the tables in Fig. 7, which represents the SMM shown in Fig. 2 and 3. The Table 7-(a) represents the functions p, r and the edges created in the Main procedure. The number of tuples in 7-(a) is n+1. The table 7-(a) requires

$(n+1)(2\log(n+1)+\log|\Sigma|)$ bits. The table 7-(b) represents the edges created in procedure CONST, i.e., start node, input symbol, interval and end node of each edge. Each edge labeled by $[n_1,n_2]$ $(n_1 \leq n_2)$ is decomposed into n_2-n_1+1 edges labeled by $[n_1,n_1]$, $[n_1+1,n_1+1]$,..., and $[n_2,n_2]$. So an interval is represented by one column in 7-(b). By the Proof of Theorem 2, it is proved that the number of tuples of the table 7-(b) never exceed n. As well as McCreight, we can use Lampson's hash function to store the table 7-(b)[7]. Then Table 7-(b) requires $n(\log|\Sigma|+3\log(n+1)+1)$ bits. String w requires $n\log|\Sigma|$ bits. Finally we have the following theorem concerning about the storage space of the SMM.

[Theorem 3] The SMM for $w(|w|=n)$ requires $(3n+1)\log|\Sigma|+(5n+2)\log(n+1)+n$ bits.

The algorithm for computing all MCSs between two strings s and $w(|s|=m,|w|=n|)$ requires $4(m+n+1)\log(m+n+1)+3(m+n+1)\log|\Sigma|+4(m+n+1)$ bits by the proposition 3. On the other hand, to compute all MCSs of s w.r.t. w, we only apply s to the SMM of w. So we only need $(3n+m)\log|\Sigma|+(5n+2)\log(n)+n$ bits. By this result, SMM is efficient at the view of the storage space for the purpose of computing MCSs.

4. **Computation** of MRSs and MCSs and their application to data compression

In 4.1, procedures to compute MRSs and MCSs are presented and in 4.2, online data compression procedures by MRSs and MCSs are briefly discussed.

4.1 Computation of MRSs and MCSs

By the definition of r and p, we can find all MRSs. The following proposition is useful for this purpose.

[Proposition 4] Consider the SMM for w. When $r(s_{i+1})\neq r(s_i)+1$, $w(i-r(s_i)+1:i)$ and $w(j-r(s_i)+1:j)$ are candidates of MRSs of w, where $s_j=p(s_i)$.

(Proof) We show an MRS of w is included among the candidates obtained in Proposition 4. Let an MRS of w be w(i-k+1:i). By the definition, neither w(i-k:i) nor w(i-k+1:i+1) is an RS of w.

(1)When w(j-k+1:j)=w(i-k+1:i) for some j(<i): Because w(i-k:i) is not an RS, w(i-k:i) is MBW_i. By the definition, $r(s_i)=k$. Because w(i+1-k:i+1) is not an RS, $r(s_{i+1})=|MBW_{i+1}|-1\leq k+1-1=k=r(s_i)$. In this case, w(j-k+1:j) satisfies the condition of Proposition 4.

(2)When w(j-k+1:j)=w(i-k+1:i) for j(>i): Let j be such the minimum number. Then $p(s_j)=s_i$ and $r(s_j)=k$ hold. So w(i-k+1:i) satisfies the condition of Proposition 4. (Q.E.D.)

By Proposition 4, we can have a linear time procedure to find all MRSs.

[Procedure 2] Calculation of MRSs

(1)Construct the SMM for a string w(|w|=n) by Procedure 1. A(i)<--0 ($1\leq i\leq n$). $r(s_{n+1})$<--0

comment: the array A stores MRSs.

(2)For i=1 to n

If $r(s_{i+1})\neq r(s_i)+1$ then begin

Let $s_j=p(s_i)$. $A(i-r(s_i)+1)$<--$r(s_i)$, $A(j-r(s_i)+1)$<--$r(s_i)$. end

end

(3) If A(1)≠0 then w(1:A(1)-1) is an MRS.

If A(1)=0 then X<-- A(1) else x<--A(1)-1.

For i=2 to n

If A(i)+i-1>X then begin w(i:i+A(i)-1) is an MRS. X<--i+A(i)-1. end

end end of Procedure 2.

For Procedure 2, next theorem holds(Proof omitted).

[Theorem 4] Procedure 2 requires time of O(|w|) and it is an off-line procedure.

We can show in 4.2 an on-line data compression procedure utilizing MRSs.

By the definition of SMM, MCSs can be computed efficiently. Before the formal description, an example is given.

[Example 6] s=a b b b a b and w=b b a a b b a b a are considered. The SMM for w is shown in Figures 2 and 3. We can find all MCSs of s w.r.t. w as follows. The machine starts from s_0 with 0 in A. When the first symbol s(1) is applied, the machine enters s_3 and A has 1. When s(2)="b" is applied, the machine enters $\delta'(s_1,b,1)$ and A has 2. By the same way, after applying s(3), the machine is in s_6 and A has 3. When s(4)="b" is applied, $\delta'(s_6,b,3)$=^(undefined). By the definition of SMM, s(1:3)(=w(4:6)) is found as an MCS of s w.r.t. w. Then we consider $\delta'(p(s_6),b,r(s_6))$. Because $\delta'(p(s_6),b,r(s_6))$=^, $\delta'(p^{(2)}(s_6),b,r^{(2)}(s_6))$ $(=s_2)$ is considered. Finally the machine enters s_2 and A has $r^{(2)}(s_6)+1=2$. By the same way, s(3:6) (=w(5:8)) is obtained.

The brief procedure to calculate all MCSs is as follows.

[Procedure 3] Calculation of MCSs of s w.r.t. w

(1) Construct the SMM for w. X<--0.

(2) For i=1 to |s|

Apply s(i) to the SMM. If A has not X+1 then s(i-X:i-1) is an MCS of s w.r.t. w. X<--the content of A. end

(3) X<--the content of A. s(|s|-X+1:|s|) is an MCS.

end of Procedure 3

The following theorem holds. Proof is omitted.

[Theorem 5] Procedure 3 requires time of O(|s|+|w|) and it is an on-line procedure.

Then we show data compression procedures.

4.2 On-line data compression procedures

The authors have presented data compression procedures utilizing MRSs and MCSs[5]. In [5], the position tree is used to compute MRSs and MCSs. In this section, we consider on-line data compression procedures.

[Definition 9] An encoded datum $\bar{w}$ is sequentially decodable if $\bar{w}$ can be decoded by scanning from the head to the tail.

It is easily proved that if a coding procedure is online, the resulting code is sequentially decodable. So there is an online decoding procedure for codes generated by our data compression procedures. The first one is to utilize MRSs. Rodeh et al. have proved that a data compression procedure utilizing RSs can give optimal encoding scheme as the length of a string grows to infinity. The key idea of utilizing MRSs is to replace the second or later occurrences of an MRS by identifiers of the first occurrence of the MRS. Consider the following example.

[Example 7] Consider a string w=b a a a a a a a b a a a . We can know w(10:13)=w(1:4) and w(3:9)=w(2:8). Then we need not store the whole string. w is expressed by w(1:2)w(3:9)w(10:13) which can be expressed by w(1:2)w(2:8)w(1:4). Corresponding to this sequence, we will define a coded string $\bar{w}$=ba#2,6#1,3, where w(i:i+k) is represented by #i,k. Note that $\bar{w}$ is sequentially decodable.

Remember that the second or later occurrences of an MRS are represented by the reset function of the SMM by Proposition 4. Then an on-line procedure is obtained. This procedure is similar to the one using MCSs which is shown below. The major difference is to use MRS instead of MCS. We omit the formal procedure in this paper.

Consider the data compression procedure by MCSs. We assume the situation when similar data are to be stored and only one of them is referred

very often. In this case, the most frequently referred datum is used as a reference datum (say it w_0). Other data are stored using w_0. We represent a datum by a concatenation of substrings of w_0. When two data are similar, one datum is represented by a concatenation of few substrings of the other. Consider the following example.

[Example 8] Consider two strings w_1=a a d c a d c and w_2=c a a d c b a. w_1(1:4) and w_1(5:7) are MCSs of w_1 w.r.t. w_2, each of which equals w_2(2:5) and w_2(3:5) respectively. When w_2 is used as a reference datum of w_1, w_1 can be expressed by a concatenation of substrings of w_2. When w_2(i:i+k) is coded as i,k, w_1 is coded as 2,2,3,3,2, where the first value 2 means that the reference datum is w_2.

For the best compression, a datum must be expressed by the concatenation of the least number of substrings of the reference datum. Following procedure is an online data compression procedure and generates the best results in this sence.

[Procedure 4] On-line data compression procedure by MCSs

Let w_0 be a reference datum. $w_1,\ldots,w_N$ are other data.

(1) Construct the SMM for w_0.

(2) For j=1 to N do

```
    X<--1.  i<--1.  h<--0.  Suppose the SMM is in s0 and A has 0.
    Do while(not end of wj)
      begin  Input wj(i).  Let the present state be sp.
        If δ'(sp,wj(i),content of A) is undefined then
          begin wj(i-r(si-1):i-1) is an MCS.
            If i-r(si-1)≦X then begin h<--i-1. m<--p. end
              else begin  replace wj(X:h) by an identifier p-(h-X),h-X.
                          X<--h+1.  h<--0.  end
                end
```

Make a transition.

end

Replace $w_j(X:|w_j|)$ by $p-(|w_j|-X),|w_j|-X$.

end end of Procedure 4

In the above procedure, the case h=0 (there are some symbols in w_j that do not appear in w_0) is not considered for simplicity. The following proposition can be proved easily[5].

[Proposition 6] Procedure 4 generates the best result in the sence that a datum is represented by the minimum number of substrings of the reference datum.

The fact that Procedure 4 may not give the optimum compression scheme because the sequential decodability is attained should be remarked.

5. Concluding Remarks

In this paper, an online and linear time construction procedure of an SMM is presented. And on-line data compression procedures utilizing the SMM are presented. The SMM seems to be more useful than previous results for many string matching problems such as finding differences between two files.

Data compression procedures by MRSs and MCSs seem to be useful for the sets of similar data. Our data compression procedures are extensions of the one presented by Kang et al.[6]. The remaining problems are i) analysis of data compression rates, ii) optimum data compression(sequentially decodability is not assumed).

[References]

[1] Aho,A.V., Hopcroft,J.E. and Ullman,J.D.: "The design and analysis of computer algorithms", Addison-Wesley, Reading Mass., 1974.

[2] Aho,A.V. and Corasick,M.J.: "Efficient string matching: An aid to bibliographic search", C.ACM, 18, 6, pp. 333-340.

[3] Findler,N.V. and Leeuwen,J.V.: "A family of similarity measures between two strings", IEEE Trans. on Pattern Analysis and Machine Intelligence, PAMI-1,1.

[4] Heckel,P.: "A technique for isolating differences between two files", C.ACM, 21, 4, 1978.

[5] Kambayashi,Y., Nakatsu,N. and Yajima,S.: "Data compression procedures utilizing similarity of strings", AFIPS NCC'81, 555-562, May 1981.

[6] Kang,A.N.C., Lee,R.C.T., Chang,C.L. and Chang,S.K.: "Storage reduction through minimal spanning trees and spanning forests", IEEE Trans. on Computer, C-26, 5, 1977.

[7] Knuth,D.E., "The art of computer programming, vol.3, sorting and searching", Addison-Wesley, Reading Mass., 1973.

[8] Knuth,D.E., Morris,J.H. and Pratt,V.R.: "Fast pattern matching in strings", SIAM J. Comput., 6, 1977.

[9] Majster,M.E. and Reiser,A.: "Efficient on-line construction and correction of position trees", SIAM J. Comput., 9, 4, 1980.

[10] McCreight,E.M.: "A space-economical suffix tree construction algorithm", JACM, 23, 2, 1976.

[11] Rodeh,M., Pratt,V.R. and Even,S.: "Linear algorithm for data compression via string matching", JACM, 28, 1, Jan. 1981.

[12] Weiner,P.: "Linear pattern matching algorithms", IEEE SWAT, 1-11, 1973.

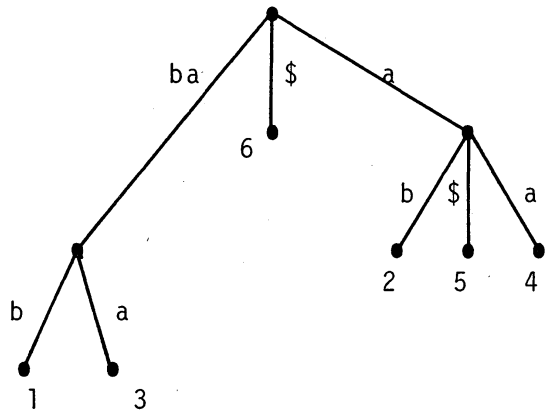


Fig. 1- The compact position tree for the string babaa.

	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9
reset fun. r	0	1	0	1	1	2	3	2	2
position fun. p	s_0	s_1	s_0	s_3	s_1	s_2	s_3	s_5	s_3

Fig. 2- The reset function and position function for the string bbaabbaba.

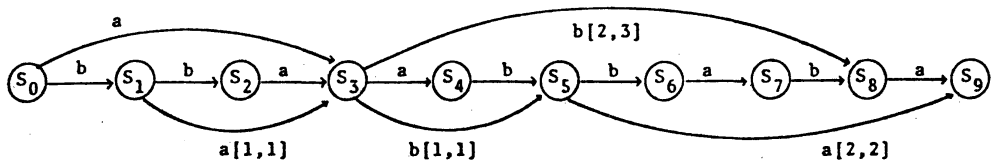


Fig. 3- **The labeled transition diagram of the SMM for the string w=b b a a b b a b a (each edge without an interval has the interval [0,∞])**

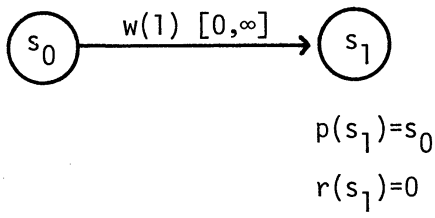


Fig. 4- The SMM for a string w(l) of length l.

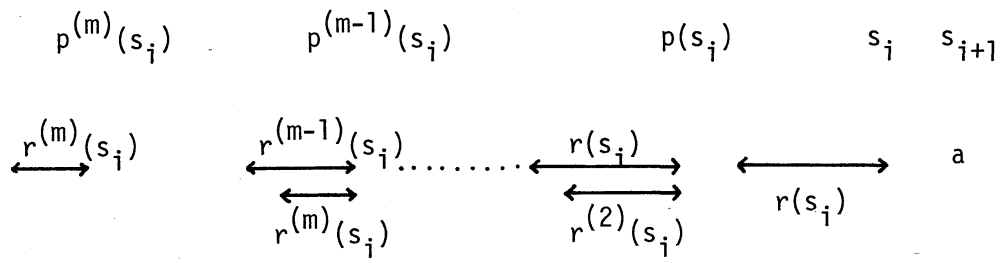
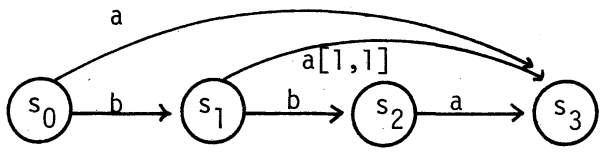


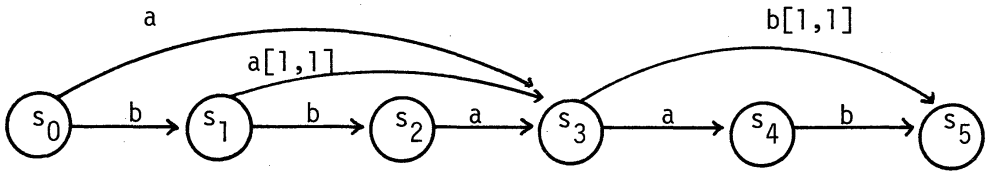
Fig. 5- Relations between substrings and two functions.

Fig. 6-a



p:	s_0	s_1	s_0
r:	0	1	0

Fig. 6-b



p:	s_0	s_1	s_0	s_3	s_1
r:	0	1	0	1	1

Fig. 6- Construction process of an SMM. (interval [0,∞] is omitted)

Table 7-(a)

P	r	input
Λ	Λ	b
s_0	0	b
s_1	1	a
s_0	0	a
s_3	1	b
s_1	1	b
s_2	2	a
s_3	3	b
s_5	2	a
s_3	2	*(any)

Table 7-(b)

state	input	interval	next state
s_0	a	∞	s_3
s_1	a	1	s_3
s_3	b	1	s_5
s_3	b	2	s_8
s_3	b	3	s_8
s_5	a	2	s_9

Fig. 7- Realization of an SMM.