# LINEAR PATTERN MATCHING ALGORITHMS

Peter Weiner

The Rand Corporation, Santa Monica, California*

Abstract

In 1970, Knuth, Pratt, and Morris [1] showed how to do basic pattern matching in linear time. Related problems, such as those discussed in [4], have previously been solved by efficient but sub-optimal algorithms. In this paper, we introduce an interesting data structure called a bi-tree. A linear time algorithm for obtaining a compacted version of a bi-tree associated with a given string is presented. With this construction as the basic tool, we indicate how to solve several pattern matching problems, including some from [4], in linear time.

## I. Introduction

In 1970, Knuth, Morris, and Pratt [1-2] showed how to match a given pattern into another given string in time proportional to the sum of the lengths of the pattern and string. Their algorithm was derived from a result of Cook [3] that the 2-way deterministic pushdown languages are recognizable on a random access machine in time O(n). Since 1970, attention has been given to several related problems in pattern matching [4-6], but the algorithms developed in these investigations usually run in time which is slightly worse than linear, for example O(n log n). It is of considerable interest to either establish that there exists a non-linear lower bound on the run time of all algorithms which solve a given pattern matching problem, or to exhibit an algorithm whose run time is of O(n).

In the following sections, we introduce an interesting data structure, called a bi-tree, and show how an efficient calculation of a bi-tree can be applied to the linear-time (and linear-space) solution of several pattern matching problems.

## II. Strings, Trees, and Bi-Trees

In this paper, both patterns and strings are finite length, fully specified sequences of symbols over a finite alphabet $\Sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_t\}$. Such a pattern of length m will be denoted as

$$P = P\langle 1\rangle\, P\langle 2\rangle \ldots P\langle m\rangle,$$

where $P\langle i\rangle$, an element of $\Sigma$, is the $i^{th}$ symbol in the sequence, and is said to be located in the $i^{th}$ position. To represent the substring of characters which begins at position i of $P$ and ends at position j, we write $P\langle i{:}j\rangle$. That is, when $i \leq j$, $P\langle i{:}j\rangle = P\langle i\rangle \ldots P\langle j\rangle$, and $P\langle i{:}j\rangle = \Lambda$, the null string, for $i > j$.

Let $\Sigma^*$ denote the set of all finite length strings over $\Sigma$. Two strings $\omega_1$ and $\omega_2$ in $\Sigma^*$ may be combined by the operation of concatenation to form a new string $\omega = \omega_1\,\omega_2$. The *reverse* of a string $P = A\langle 1\rangle \ldots A\langle m\rangle$ is the string $P^r = A\langle m\rangle \ldots A\langle 1\rangle$.

The *length* of a string or pattern, denoted by $lg(\omega)$ for $\omega \in \Sigma^*$, is the number of symbols in the sequence. For example, $lg(P\langle i{:}j\rangle) = j-i+1$ if $i \leq j$ and is 0 if $i > j$.

Informally, a bi-tree over $\Sigma$ can be thought of as two related t-ary trees sharing a common node set.

Before giving a formal definition of a bi-tree, we review basic definitions and terminology concerning t-ary trees. (See Knuth [7] for further details.)

A *t-ary tree* T over $\Sigma = \{\sigma_1, \ldots, \sigma_t\}$ is a set of nodes N which is either empty or consists of a *root*, $n_0 \in N$, and t ordered, disjoint t-ary trees.

Clearly, every node $n_i \in N$ is the root of some t-ary tree $T^i$ which itself consists of $n_i$ and t ordered, disjoint t-ary trees, say $T^i_1, T^i_2, \ldots, T^i_t$. We call the tree $T^i_j$ a *sub-tree* of $T^i$; also, all sub-trees of $T^i_j$ are considered to be sub-trees of $T^i$. It is natural to associate with a tree T a *successor function*

$$S\colon N\times\Sigma \rightarrow (N-\{n_0\}) \cup \{NIL\}$$

defined for all $n_i \in N$ and $\sigma_j \in \Sigma$ by

$$S(n_i,\sigma_j) = \begin{cases} n^i, \text{ the root of } T^i_j \text{ if } T^i_j \text{ is non-empty} \\ NIL \text{ if } T^i_j \text{ is empty.} \end{cases}$$

It is easily seen that this function completely determines a t-ary tree and we write $T = (N, n_0, S)$.

If $n' = S(n,\sigma)$, we say that n and n' are connected by a *branch* from n to n' which has a *label* of $\sigma$. We call n' a *son* of n, and n the *father* of n'. The *degree* of a node n is the number of sons of that node, that is, the number of distinct $\sigma$ for which $S(n,\sigma) \neq NIL$. A node of degree 0 is a *leaf* of the tree.

It is useful to extend the domain of S from $N\times\Sigma$ to $(N \cup \{NIL\}) \times \Sigma^*$ (and extend the range to include $n_0$) by the inductive definition

(S1) $S(NIL,\omega) = NIL$ for all $\omega \in \Sigma^*$

(S2) $S(n,\Lambda) = n$ for all $n \in N$

(S3) $S(n,\omega\sigma) = S(S(n,\omega),\sigma)$ for all $n \in N$, $\omega \in \Sigma^*$, and $\sigma \in \Sigma$.

Not every $S\colon N\times\Sigma \rightarrow (N-\{n_0\}) \cup \{NIL\}$ is the successor function of a t-ary tree. But a necessary and sufficient condition for S to be a successor function of some (unique, if it exists) t-ary tree can be expressed in terms of the extended S. Namely, that there exists exactly one choice of $\omega$ such that $S(n_0,\omega) = n$ for every $n \in N$. When there exists a T such that $T = (N,n_0,S)$, we say that S is *legitimate*.

We may also associate with T a *father function* $F\colon N \rightarrow N$ defined by $F(n_0) = n_0$ and for $n' \in N-\{n_0\}$,

$$F(n') = n \Leftrightarrow S(n,\sigma) = n' \text{ for some } \sigma \in \Sigma.$$

*This work was partially supported by grants from the Alfred P. Sloan Foundation and the Exxon Education Foundation. P. Weiner was at Yale University when this work was done.

Let $F^0(n) \equiv n$ for all $n \in N$. It may be shown that the k-fold composition of F, $F^k$, for positive k and $n \neq n_0$, satisfies $F^k(n) \neq n$, and that for any n there exists a lease value of k such that $F^k(n) = n_0$. This value is called the *level* of the node. Any $n' = F^k(n)$ for positive k is said to be an *ancestor* of n. (The root $n_0$ is an ancestor of all other nodes in the tree.)

There is another important function which may be associated with a t-ary tree T over the alphabet Σ. This function $W: N \rightarrow \Sigma^*$ associates a string of symbols from Σ with each node of T, and is defined recursively by

$$\text{(W1)}\quad W(n_0) = \Lambda$$

$$\text{(W2)}\quad W(n) = W(n')\cdot\sigma \Leftrightarrow n = S(n',\sigma).$$

It is not hard to show that (W1) and (W2) completely specify a well-defined function, and moreover that the sequence of branches which connect the root to any other node n in T are labelled with the elements of W(n). (The label of the branch from $n_0$ is the leftmost element of W(n), etc.) It is also possible to show that the length of W(n) equals the level of node n. Indeed, an inductive argument can be made to establish the useful assertion that for all $n \in N$ and $\omega \in \Sigma^*$,

$$\omega = W(n) \Leftrightarrow n = S(n_0,\omega) \tag{1}$$

as well as the identity

$$n_0 = F^{lg(W(n))}(n) \text{ for all } n \in N.$$

Note also that when S is not legitimate the function W defined recursively in terms of S by (W1) and (W2) is not well defined. Thus, $(N,n_0,S)$ is a t-ary tree if and only if W is well defined. We call the function W the *walk function* associated with T.

The association of a node n with the string $\omega = W(n)$ is an important one. In order to be able to associate ω with n directly we adopt the following notational convention. If n' is a node in N, we write $\omega' = W(n')$. Similarly, write $\hat{\omega} = W(\hat{n})$ for $\hat{n} \in N$, $\ddot{\omega}' = W(\ddot{n}')$ for $\ddot{n}' \in N$, etc.

*Definition:*

A *bi-tree* $B = (N,n_0,S_p,S_s)$ over the alphabet $\Sigma = \{\sigma_1,\ldots,\sigma_t\}$ is a set of nodes N with a designated *root* $n_0 \in N$, together with the functions

$$S_p: N\times\Sigma \rightarrow (N-\{n_0\}) \cup \{NIL\}$$

and

$$S_s: N\times\Sigma \rightarrow (N-\{n_0\}) \cup \{NIL\}$$

such that

(B1) $T_p = (N,n_0,S_p)$ is a t-ary tree,

(B2) $T_s = (N,n_0,S_s)$ is a t-ary tree, and

(B3) $W_p(n) = [W_s(n)]^r$ for all $n \in N$,

where $W_p$ ($W_s$) is the walk function associated with the tree $T_p$ ($T_s$), and $[W_s(n)]^r$ is the reverse of $W_s(n)$.

We call the tree $T_p$ the p-tree associated with B, and the tree $T_s$ the s-tree associated with B. We also say that the bi-tree B is an s-extension (p-extension) of its p-tree (s-tree). When appropriate to prevent confusion, we use terms such as p-branch to indicate a branch of the p-tree, etc.; also, the function $F_p$ ($F_s$) is the father function of the tree $T_p$ ($T_s$). However, if a term or function is written without an s or p identifier, we mean to refer to the p-tree concept.

*Remark:*

It follows from the definition of a bi-tree that if a node is at the $j^{th}$ level of the p-tree it must also be at the $j^{th}$ level of the s-tree, and vice-versa. Actually, the p-tree and s-tree are anti-isomorphic images of one another in the sense of [4].

The definition of a bi-tree does not in itself insure that there exist any bi-trees at all; however, an example of a bi-tree is shown in Figure 1, which establishes that the definition is non-vacuous.

A useful relationship between the extended functions $S_p$ and $S_s$ of a bi-tree is provided in the following lemma.

*Lemma 1:*

Let $B = (N,n_0,S_p,S_s)$ be a bi-tree over Σ. Then for all $n \in N$ and $\omega \in \Sigma^*$,

$$n = S_p(n_0,\omega) \Leftrightarrow n = S_s(n_0,\omega^r).$$

*Proof:*

Consider the string $\omega = W_p(n)$. Since $(N,n_0,S_p)$ is a t-ary tree,

$$\omega = W_p(n) \Leftrightarrow n = S_p(n_0,\omega).$$

Similarly, since $(N,n_0,S_s)$ is a t-ary tree,

$$\omega = W_s(n) \Leftrightarrow n = S_s(n_0,\omega).$$

We also have, from the definition of a bi-tree, that

$$W_p(n) = [W_s(n)]^r.$$

It follows that

$$\begin{aligned} n = S_p(n_0,\omega) &\Leftrightarrow \omega = W_p(n) \\ &\Leftrightarrow \omega^r = W_s(n) \\ &\Leftrightarrow n = S_s(n_0,\omega^r). \end{aligned}$$

QED

If T is a given t-ary tree, there may or may not exist a bi-tree which is either an s-extension or p-extension of T. (Of course, the symmetry of the definition implies that if T has an s-extension B then it also has a p-extension B', and vice-versa.)

*Theorem 1:*

A given t-ary tree $T = (N,n_0,S)$ is the p-tree associated with some bi-tree B is and only if all $n \in N$, $\sigma \in \Sigma$, and $\omega \in \Sigma^*$,

$$n = S(n_0,\sigma\omega) \Rightarrow \text{there exists a } n' \in N \text{ such that } n' = S(n_0,\omega). \tag{2}$$

*Proof:*

Suppose that T is the p-tree associated with the bi-tree $B = (N,n_0,S_p,S_s)$, so that $T = (N,n_0,S_p)$. It follows from Lemma 1 that if $n = S_p(n_0,\sigma\omega)$ then $n = S_s(n_0,\omega^r\sigma)$. Consider the node $n' = F_s(n) = S_s(n_0,\omega^r)$. Lemma 1 also implies that $n' = S_p(n_0,\omega)$, and (2) is established.

Now assume that (2) holds. Let W denote the walk function of T, and define $B = (N,n_0,S_p,S_s)$ by $S_p = S$ and for all $n \in N$ and $\sigma$ in $\Sigma$,

$$S_s(n,\sigma) = S_p(n_0,\sigma W_p(n)). \qquad (3)$$

We must establish that (B1), (B2), and (B3) hold for B. Certainly, $T_p = (N,n_0,S_p)$ is a t-ary tree. To show that $T_s = (N,n_0,S_s)$ is a t-ary tree, and to establish (B3), we prove by induction that the function $W_s$ defined inductively in terms of $S_s$ by (W1) and (W2) above is well defined and satisfies

$$\text{(B3)}\ [W_s(n)]^r = W_p(n) \text{ for all } n \in N.$$

The induction is on the length of $W_p(n)$. If $lg(W_p(n)) = 0$, then $n = n_0$, and from (W1) we have $[W_s(n_0)]^r = \Lambda = W_p(n_0)$. Also, since the range of $S_s$ does not include $n_0$, the value of $W_s(n_0)$ is well defined by (W1) and (W2).

The inductive hypothesis is that if n' is any node with $lg(W_p(n')) = k$, then (B3) holds for this node, and that the value of $W_s$ obtained from (3), (W1) and (W2) is well defined. Let n be a node such that $W_p(n) = \sigma\omega$, where $lg(\omega) = k$. From (1) we have that $n = S_p(n_0,\sigma\omega)$, so (2) implies that there exists a (unique!) $n' \in N$ such that $n' = S_p(n_0,\omega)$. Using (1) once more, we obtain $\omega = W_p(n')$. Thus, from (3), $S_s(n',\sigma) = S_p(n_0,\sigma\omega) = n$. The inductive hypothesis allows that $W_s(n')$ is well defined, and since (W2) defines $W_s(n)$ in terms of $W_s(n')$ we can see that $W_s(n)$ is also well defined. The inductive hypothesis also establishes that $W_s(n') = [W_p(n')]^r = \omega^r$. Finally, we may deduce that $W_s(n) = W_s(n')\sigma = \omega^r\sigma = (\sigma\omega)^r = [W_p(n)]^r$. This completes the induction and the proof.

QED

We now relate the concept of a bi-tree to that of a string. First, however, consider the basic problem of finding a match of a given pattern $P$ of length $\ell$ with another string $S$ of length $\ell'$, where $\ell' \geq \ell$. That is, find positions i and j within $S$ such that $P = S\langle i:j\rangle = S\langle i\rangle S\langle i+1\rangle \ldots S\langle j\rangle$. Clearly, if $P$ does match some substring of $S$, then $P^r = P\langle \ell\rangle \ldots P\langle 1\rangle$, the reverse of $P$, also matches a substring of $S^r$. This observation implies that every technique which solves a pattern matching problem working from left to right has a dual procedure which works from right to left. In what follows, we adopt a left to right viewpoint, referring only briefly to dual concepts as appropriate. With this understanding, we henceforth assume (for purely technical reasons) that every string $S \in \Sigma^*$ ends in a symbol which does not occur elsewhere in $S$. Also, when we refer to the substring located at position i of $S$, we mean that $S\langle i\rangle$ is the leftmost symbol of the substring.

*Definition:*

The *prefix-tree associated with string S* over $\Sigma = \{\sigma_1,\ldots,\sigma_t\}$ is a t-ary tree $T_p = (N,n_0,S_p)$ with exactly $lg(S)$ leaves such that there is a bijective *pointer function* J from the set of leaves of $T_p$ to the set of positions within $S$ such that if $j = J(n)$ then $W_p(n)$ is the minimal length unique substring of $S$ whose leftmost symbol is located at position j of $S$. That is, $W_p(n) = S\langle j:k\rangle$ occurs only once in $S$ and $S\langle j:k-1\rangle$ occurs at least twice in $S$. We call $I(j) = S\langle j:k\rangle$ the *prefix identifier* associated with position j.

(The concept of suffix tree may be similarly defined for strings with left endmarkers.)

The assumption that $S$ has a unique endmarker on the right insures that every position of $S$ has a prefix identifier. This implies that there is exactly one prefix-tree associated with a given $S$. Moreover, if n is any node of the prefix-tree of $S$, then $W_p(n)$ is a substring of $S$. Indeed, the substring $W_p(n)$ occurs at every position $J(n')$ of $S$ such that n' is a leaf of the sub-tree whose root is n. Consequently, the minimality condition concerning $W_p(n)$ implies that the index of any father of a leaf is greater than one, and that every leaf node has a brother.

Figure 2 shows the prefix-tree associated with the string $S = 011010\vdash$.

As may be surmised from our choice of terminology, the prefix-tree associated with any string $S$ has an s-extension.

*Theorem 2:*

For every string $S$ over $\Sigma$, there exists a bi-tree of $S$, $B^p = (N,n_0,S_p,S_s)$ such that $(N,n_0,S_p)$ is the prefix-tree associated with $S$.

*Proof:*

Consider the prefix tree $T_p = (N,n_0,S_p)$ associated with $S$. We first show that if node $n \in N$ is equal to $S(n_0,\sigma\omega)$ for some $\sigma \in \Sigma$, $\omega \in \Sigma^*$, then there exists a node $n' = S_p(n_0,\omega)$ in N. The assumption that $n = S_p(n_0,\sigma\omega)$ implies that $W_p(n) = \sigma\omega$, so $\sigma\omega$ must be a substring of $S$. Moreover, either $\sigma\omega$ occurs at least twice in $S$, or $\sigma\omega$ is a prefix identifier of $S$. If $\sigma\omega$ occurs more than once, so must $\omega$; if $\sigma\omega$ is a prefix identifier, then either $\omega$ is a prefix identifier or $\omega$ occurs more than once, since every prefix of $\omega$ must occur more than once. In either case, there is a node n' with $W_p(n') = \omega$ and $S_p(n_0,\omega) = n'$. Theorem 1 can now be directly invoked to complete the proof.

QED

From the proof of Theorem 1, recall that the s-tree $T_s = (N,n_0,S_s)$ of $B^p$ is defined by (3). We call the bi-tree $B^p$ the *prefix bi-tree* associated with $S$. (It is also true that there exists a *suffix bi-tree* associated with every string $S$ with a left endmarker.)

As will be shown in Section IV, linear-time and linear-space algorithms for certain pattern matching problems can be derived assuming that an appropriate prefix-tree is pre-calculated. We have been unable to find efficient methods for directly obtaining a prefix-tree. But as we show in the next section, efficient methods exist for calculating the prefix bi-tree whose p-tree is the desired prefix-tree; more important, a linear-time, linear-space algorithm for obtaining a *compacted prefix bi-tree* will be exhibited.

## III. Computation of Prefix Bi-Trees and Compacted Prefix Bi-Trees

It is well to consider first a direct method for obtaining the prefix-tree associated with a given string $S$ of length m.

Our direct method is an iteration of an algorithm to compute the prefix-tree $T_i$ of the suffix substring $S_i = S\langle i{:}m\rangle$ assuming that the prefix-tree $T_{i+1}$ of the suffix substring $S_{i+1} = S\langle i+1{:}m\rangle$ is known. The following lemma provides the theory which both motivates the algorithm and which can be used to prove its correctness. Its usefulness in this regard is based on the observation that the prefix-tree T of a string $S$ is completely determined by the set $I = \{I(j) | 1 \leq j \leq m\}$ of prefix identifiers.

*Lemma 2:*

Let $I_{i+1} = \{I_{i+1}(j) | i < j \leq m\}$ be the set of prefix identifiers of $S_{i+1} = S\langle i+1{:}m\rangle$ and let $I_i$ be the set of prefix identifiers of $S_i$. Define $\tilde{\omega}$ to be the longest prefix of $S_i$ which is also a prefix of some element of $I_{i+1}$. If $\tilde{\omega} \neq I_{i+1}(j_0)$ for some $j_0$, then $I_i(j) = I_{i+1}(j)$ when $i < j \leq m$ and $I_i(i) = \tilde{\omega}\, S\langle i+lg(\tilde{\omega})\rangle$. Otherwise, if $\tilde{\omega} = I_{i+1}(j_0)$, then $I_i(j) = I_{i+1}(j)$ when $i < j \leq m$ and $j \neq j_0$, $I_i(j_0) = \hat{\omega} = S\langle j_0{:}j_0+lg(\hat{\omega})-1\rangle$ and $I_i(i) = S\langle i{:}i+lg(\hat{\omega})-1\rangle$, where $\hat{\omega}$ is the shortest prefix of $S_{j_0}$ which does not equal a prefix of $S_i$.

*Proof:*

Suppose first that $\tilde{\omega}$ is not an element of $I_{i+1}$, and consider the prefix of $S_i$ given by $\tilde{\omega}\, S\langle i+lg(\tilde{\omega})\rangle$. This string is the shortest prefix of $S_i$ which is not a prefix of any element of $I_{i+1}$; conversely, no element of $I_{i+1}$ is a prefix of this string. Thus the prefix identifiers of $S_i$ are given by $I_i(j) = I_{i+1}(j)$ for $i < j \leq m$ and $I_i(i) = \tilde{\omega}\, S\langle i+lg(\tilde{\omega})\rangle$.

Now suppose that $\tilde{\omega} = I_{i+1}(j_0)$. Surely, $\tilde{\omega}$ is not a prefix of any other member of $I_{i+1}$, nor can any member of $I_{i+1}$ other than $I_{i+1}(j_0)$ be a prefix of $\tilde{\omega}$. This insures that for $i < j \leq m$ and $j \neq j_0$, $I_i(j) = I_{i+1}(j)$. $I_i(j_0)$ and $I_i(i)$ must each be strings which are of length one greater than the length of their largest common prefix, say $\ddot{\omega}$. Thus, $I_i(j_0) = \hat{\omega} = \ddot{\omega}\, S\langle j_0+lg(\ddot{\omega})\rangle = S\langle j_0{:}j_0+lg(\hat{\omega})-1\rangle$ and $I_i(i) = \ddot{\omega}\, S\langle i+lg(\ddot{\omega})\rangle = S\langle i{:}i+lg(\hat{\omega})-1\rangle$. Clearly, $\hat{\omega}$ is the shortest prefix of $S_j$ which does not equal a prefix of $S_i$.

QED

The two cases described in the proof of Lemma 2 have interesting analogies in the tree $T_{i+1}$. In the first case $\tilde{n}$ is an internal node and $T_i$ may be formed by adding the leaf $n_i$ (with $J(n_i) = i$) to $T_{i+1}$ by connecting it to $\tilde{n}$ with a branch labelled $S\langle i+lg(\tilde{\omega})\rangle$. In the second case, $\tilde{n}$ is the leaf with $J(\tilde{n}) = j_0$. Here, $\tilde{n}$ must be replaced by a two-leaf subtree rooted at $\tilde{n}$. The node $\hat{n}$ is the leaf with $J(\hat{n}) = j_0$. The other leaf of the subtree, $n_i$, has $J(n_i) = i$. These two nodes, as well as any other nodes in the subtree between the root $\tilde{n}$ and the two leaves must also be added to $T_{i+1}$ to form $T_i$. In the first case, we say that $T_i$ is obtained from $T_{i+1}$ by a *type 1* construction; in the second case, by a *type 2* construction. It is also useful to distinguish three subcases of a type 2 construction: 2a) $\tilde{n} = \ddot{n}$ is the father of $\hat{n}$, 2b) $\tilde{n}$ is the father of $\ddot{n}$ (the father of $\hat{n}$), and 2c) $\tilde{n}$ is an ancestor (but not the father) of $\ddot{n}$.

Figure 3 illustrates these cases and our notational conventions.

In all cases, the calculation of $T_i$ from $T_{i+1}$ suggested by the Lemma first locates the node $\tilde{n}$. Algorithm D, below, implements this calculation by walking $T_{i+1}$ from the root $n_0$ by traversing the branches which are labelled with symbols from the prefix of $S_i$.

We assume that at the beginning of the algorithm $T_{i+1}$ is available as a set of nodes $N_{i+1}$ and a function S defined for $N_{i+1} \times \Sigma$. The construction of $T_i$ forms $N_i$ by adding nodes to $N_{i+1}$. (When a new node is added, we assume that all unspecified values of S, etc., are initialized to NIL.) It is also convenient to associate with each node n a label $J_n$ which is the position in $S_{i+1}$ of the (rightmost, if not unique) substring $\omega = W(n)$. (Note that $J_n$ is consistent with the pointer function J(n) in that if n is a leaf, $J_n = J(n)$.)

<u>Algorithm D</u> (Direct construction of $T_i$ from $T_{i+1}$)

D1. [Initialize] Set $\tilde{n} \leftarrow n_0$ and $k \leftarrow 0$.

D2. [Find $\tilde{n}$] If $S(\tilde{n}, S\langle i+k\rangle) = $ NIL go to D3; otherwise, set $\tilde{n} \leftarrow S(\tilde{n}, S\langle i+k\rangle)$, increment k, and repeat step D2. (When D3 is entered, $\tilde{\omega} = W(\tilde{n})$.)

D3. [Is $\tilde{n}$ a leaf?] Set $\ddot{n} \leftarrow \tilde{n}$. If $S(\ddot{n}, S\langle J_n+k\rangle) \neq$ NIL go to D6. (This is the case that $\tilde{\omega}$ is not a prefix identifier of $T_{i+1}$.)

D4. [Move leaf] Add node $\hat{n}$ and connect to $\ddot{n}$ by setting $S(\ddot{n}, S\langle J_{\ddot{n}}+k\rangle) \leftarrow n$. Label the leaf by setting $J_{\hat{n}} \leftarrow J_{\ddot{n}}$.

D5. [Compare symbols of $S$] If $S\langle i+k\rangle$ equals $S\langle J_n+k\rangle$ set $\ddot{n} \leftarrow \hat{n}$, increment k, and go to D4.

D6. [Add $n_i$ to tree] Add node $n_i$ and connect to $\ddot{n}$ by setting $S(\ddot{n}, S\langle i+k\rangle) \leftarrow n_i$. Label the leaf by setting $J_{n_i} \leftarrow i$. Stop. (At this point $\ddot{n}$ is the father of $n_i$.)

The prefix-tree of $S$ can be obtained by successively deriving the prefix-trees of $S_m, S_{m-1}, \ldots, S_2$, $S_1 = S$. Figure 4 shows several steps in a typical calculation.

Note that at each iteration as many as $O(lg(S_i))$ steps may be required. There are m iterations, and it may easily be shown that the total number of steps can be of $O(m^2)$. What's worse, it may be seen that this algorithm is at best $O(m \log m)$.

We now turn to the problem of finding the prefix bi-tree $B_i^p$ corresponding to $S_i$ given the prefix bi-tree $B_{i+1}^p$. Our method is based on a relationship between $I_{i+1}(i+1)$ and $I_i(i)$ which is described in the following lemma.

*Lemma 3:*

Suppose that the symbol $S\langle i\rangle$ occurs within the string $S_{i+1}$, and let $\ddot{\omega}'$ be the longest prefix of $I_{i+1}(i+1)$ which occurs at some position other than i+1, say $j_0+1$, such that $S\langle j_0\rangle = S\langle i\rangle$. Then $I_i(i) = S\langle i\rangle\ddot{\omega}'S\langle i+lg(\ddot{\omega}')+1\rangle$.

*Proof:*

Since $I_{i+1}(i+1)$ is a prefix identifier of $S_{i+1}$, it can occur only once in $S_{i+1}$ and clearly $S\langle i\rangle I_{i+1}(i+1)$ also occurs exactly once in $S_i$. Thus $I_i(i)$ is of the form $S\langle i\rangle\omega_i'$, where $\omega_i'$ is a prefix of $I_{i+1}(i+1)$. Since we have assumed that $S\langle i\rangle$ occurs within $S_{i+1}$, $lg(\omega_i') \geqq 1$, and we can write $\omega_i' = \ddot{\omega}'S\langle i+lg(\ddot{\omega}')+1\rangle$ and be sure that $\ddot{\omega}'$ occurs at least twice† in $S_{i+1}$. This implies that $\ddot{\omega}'$ is the longest prefix of $I_{i+1}(i+1)$ which occurs at some position other than i+1, say $j_0+1$, such that $S\langle j_0\rangle = S\langle i\rangle$. We have just established that $I_i(i) = S\langle i\rangle\omega_i' = S\langle i\rangle\ddot{\omega}'S\langle i+lg(\ddot{\omega}')+1\rangle$.

QED

In those situations where $S\langle i\rangle$ does not occur within $S_{i+1}$, it is trivial to show that $I_i(i) = S\langle i\rangle$.

We now wish to draw out some important relationships between the strings defined in the preceding lemmas and proofs.

In the proof of Lemma 2, we have used $\ddot{n}$ to represent the father node of $n_i$, where $\omega_i = I_i(i) = W_p(n_i)$ in $B_i^p$. It follows from Lemma 3 that $\ddot{\omega} = S\langle i\rangle\ddot{\omega}'$, where $\ddot{\omega}'$ is the longest prefix of $\omega_{i+1} = I_{i+1}(i+1)$ which occurs elsewhere in $S_{i+1}$ with $S\langle i\rangle$ to its left. This relationship implies that $\ddot{n}$ and $\ddot{n}'$ are related by $\ddot{n} = S_s(\ddot{n}', S\langle i\rangle)$ in the prefix bi-tree $B_i^p$. Similarly, if $\hat{n}' = S(\ddot{n}',\sigma)$, where $\sigma = S\langle j_0+lg(\ddot{\omega}')+1\rangle$, it may be seen that $\hat{n} = S_s(\hat{n}', S\langle i\rangle)$. Indeed, all ancestors of $\hat{n}$ except for $n_0$ are s-sons of ancestors of $\hat{n}'$ with branches labelled by $S\langle i\rangle$. In particular, the node $\tilde{n}'$ which is the closest ancestor of $\ddot{n}'$ having a non-NIL s-son labelled $S\langle i\rangle$ in $B_{i+1}^p$ plays an important role in forming $B_i^p$, since all ancestors of $\hat{n}'$ which are descendants of $\tilde{n}'$ will be s-fathers of nodes in $B_i^p$ that do not exist in $B_{i+1}^p$. Note that the situation $\tilde{n}' = \hat{n}'$ corresponds to the condition $\omega \neq I_{i+1}(j_0)$, and $n_i$ is the only node in $B_i^p$ which is not in $B_{i+1}^p$. (This requires a type 1 construction.) Also, when $\tilde{n}'$ is an ancestor of $\hat{n}'$, a sub-tree with leaves $n_i$ and $\hat{n}$ must be added to $B_{i+1}^p$ to obtain $B_i^p$. (This requires a type 2 construction.) This sub-tree is rooted at node $\tilde{n}$ and $W(\tilde{n}) = \tilde{\omega} = I_{i+1}(j_0)$. In all cases, $n_i = S_s(n_i', S\langle i\rangle)$, where $n_i' = S(\ddot{n}',\sigma)$ for $\sigma = S\langle i+lg(\ddot{\omega}')+1\rangle$. Figure 5 illustrates the notation by showing part of the prefix bi-trees $B_{i+1}^p$ and $B_i^p$.

---

†By convention, we assume that the empty string, $\Lambda$, occurs at every position of $S_{i+1}$.

Lemma 3 and the relationships just described suggest that each node of the prefix bi-tree $B_{i+1}^p$ be labelled with a t-long vector $L_n$. The $j^{th}$ component $L_n(j)$, $1 \leqq j \leqq t$, is set equal to NIL if the substring $\sigma_j W(n)$ does not occur within $S_{i+1}$; if $\sigma_j W(n)$ does occur within $S_{i+1}$, then $L_n(j)$ is the (rightmost, if not unique) position of $S_{i+1}$ such that $\sigma_j W(n)$ is a prefix of $S_{L_n(j)-1}$. Note that $J_n$ is now simply the largest non-NIL component of $L_n$, unless $J_n = i+1$. Note also that if position $j_0$ is contained in the vector $L_n$, then some p-leaf of the p-sub-tree rooted at n corresponds to the prefix identifier located at position $j_0$.

Algorithm B, below, calculates a labelled $B_i^p$ from a labelled $B_{i+1}^p$. It begins by walking from node $n_{i+1}$ towards the root until the node $\ddot{n}'$ is reached. This node is identified by the fact that it is the first node reached with a non-NIL $L_{\ddot{n}'}(j)$, where $S\langle i\rangle = \sigma_j$. The node below $\ddot{n}'$ on the path from $n_{i+1}$ is the node $n_i'$ and $n_i$ will eventually be installed as the s-son under $\sigma_j$ of $n_i'$ and as a p-son of $\ddot{n}$. Also, if node $\hat{n}$ is not already present (as determined by examining the s-branch of $\ddot{n}'$ labelled $S\langle i\rangle$) a sub-tree rooted at $\tilde{n}$ is added to $B_{i+1}^p$ to obtain $B_i^p$. The node $\tilde{n}$, which is the nearest ancestor of $\hat{n}$ already in $B_{i+1}^p$, is located by walking from $\ddot{n}'$ towards the root until a node $\tilde{n}'$ is reached with a non-NIL s-branch labelled $S\langle i\rangle$. The node $\tilde{n}$ is the s-son of $\tilde{n}'$ just found. It should now be clear that the primary role of the s-tree is to determine efficiently whether a node required in $B_i^p$ is already present in $B_{i+1}^p$.

All of the functions and labels used in Algorithm D are also used in Algorithm B. In addition, we use the vector label $L_n$ as discussed, as well as the p-father function F, the s-successor function $S_s$, and a label $LG_n$ which gives the value of $lg(W(n))$.

<u>Algorithm B</u> (Efficient construction of $B_i^p$ from $B_{i+1}^p$)

B1. [Initialize] Set $\ddot{n}' \leftarrow n_{i+1}$ and set j to the index of $S\langle i\rangle$. ($W(n_{i+1}) = I_{i+1}(i+1)$ and $S\langle i\rangle = \sigma_j$.)

B2. [Check label] If $L_{\ddot{n}'}(j)$ = NIL go to step B3; otherwise go to step B4. (When B4 is entered, $\ddot{\omega}' = W(\ddot{n}')$.)

B3. [Label and walk] Set $L_{\ddot{n}'}(j) \leftarrow i+1$ and $n_i' \leftarrow \ddot{n}'$. If $n_i' \neq n_0$, set $\ddot{n}' \leftarrow F(n_i')$ and go to step B2; otherwise set $\ddot{n} \leftarrow n_0$ and go to step B7.

B4. [Find $\hat{n}'$] Set $\hat{n} \leftarrow S(\ddot{n}', S\langle L_{\ddot{n}'}(j)+LG_{\ddot{n}'}\rangle)$.

B5. [Is $\hat{n}$ in $B_{i+1}^p$?] If $S_s(\hat{n}',\sigma_j) \neq$ NIL, set $\ddot{n} \leftarrow S_s(\ddot{n}',\sigma_j)$ and go to step B7. (This is the case when $\tilde{\omega}$ is not a prefix identifier of $S_{i+1}$.)

B6. [Add $\hat{n}$ and ancestors] If $S_s(F(\hat{n}'),\sigma_j)$ = NIL, use an implicit pushdown stack to save $\hat{n}'$ and repeat this step *recursively* with $\hat{n}'$ equal to $F(\hat{n}')$. (When this point is reached, $S_s(F(\hat{n}'),\sigma_j) \neq$ NIL.) Set $\ddot{n} \leftarrow S_s(F(\hat{n}'),\sigma_j)$ add node $\hat{n}$ by setting

$S(\ddot{n}, S\langle J_{\ddot{n}}+LG_{\ddot{n}}\rangle) \leftarrow \hat{n}$, $F(\hat{n}) \leftarrow \ddot{n}$, and $S_s(\hat{n}', \sigma_j) \leftarrow \hat{n}$. Label $\hat{n}$ by setting $J_{\hat{n}} \leftarrow J_{\ddot{n}}$, $LG_{\hat{n}} \leftarrow LG_{\hat{n}'}+1$, and $L_{\hat{n}}(k) \leftarrow L_{\ddot{n}}(k)$ for $1 \leqq k \leqq t$.

B7. [Add $n_i$] Add node $n_i$ by setting $S(\ddot{n}, S\langle i+LG_{\ddot{n}}\rangle) \leftarrow n_i$, $F(n_i) \leftarrow \ddot{n}$, and $S_s(n_i', \sigma_j) \leftarrow n_i$. Label $n_i$ by setting $J_{n_i} \leftarrow i$, $LG_{n_i} \leftarrow LG_{n_i'}+1$, and $L_{n_i}(k) \leftarrow$ NIL for $1 \leqq k \leqq t$. Stop.

As with our direct method, the prefix bi-tree of $S$, $B^P$, can be obtained by successively calculating $B^P_m, B^P_{m-1}, \ldots, B^P_1 = B^P$. We now show that the total number of operations in this process is $O(k)$, where k is the number of nodes in $B^P$. Notice that every time Algorithm B is executed, a constant number of operations is performed, except in steps B2, B3, and B6, which may be repeated several times. However, every time these steps are executed, labels are added to the tree, and these labels are never modified. It follows, since there are only $O(k)$ possible labels, that the total number of operations is also of $O(k)$. Unfortunately, Figure 6 shows a string whose prefix-tree has $O(n^2)$ nodes. Thus, while we have certainly described an efficient method for finding prefix bi-trees, this is not directly useful in obtaining linear pattern matching algorithms.

In order to overcome the difficulties associated with the large number of nodes possible in a prefix-tree, we introduce a structure called a compacted prefix-tree.

*Definition:*

Let $T = (N, n_0, S)$ be the prefix-tree associated with string $S$. The *compacted prefix-tree* of $S$ is a structure $T^c = (N^c, n_0, S^c)$, where $N^c \subseteq N$ is specified by

$$n \in N^c \Leftrightarrow \begin{array}{l}\text{the degree of } n \text{ in } T \text{ is at least two, or}\\ \text{the degree of } F(n) \text{ in } T \text{ is at least two.}\end{array}$$

For every $n' \in N^c$ and $\sigma$ such that $S(n', \sigma) \neq$ NIL, let $\omega' = \sigma\omega''$ be the shortest substring such that $S(n', \sigma\omega'') = n'' \in N^c$. We define $S^c$, a function from $N^c \times \Sigma$ to $(N^c - \{n_0\}) \cup \{\text{NIL}\}$ by

$$S^c(n', \sigma) = \begin{cases}\text{NIL} & \text{if } S(n', \sigma) = \text{NIL}\\ S(n', \sigma\omega'') & \text{otherwise.}\end{cases}$$

Observe that every internal (non-leaf) node in $N^c$ with degree one has as its only son a node of degree two or more. It is easy to show that every t-ary tree with k leaves that does not contain any internal nodes of degree one has at most k-1 internal nodes (see [7], pages 399-404). From this fact, it follows that the number of nodes in a compacted prefix-tree associated with a string of length m cannot exceed 2(m-1)+m = 3m-2. Thus, size considerations alone do not rule out the possibility of a linear algorithm to compute $T^c$. But as with non-compacted prefix-trees, we find it useful to compute instead a related compacted prefix bi-tree.

*Definition:*

Let $B^P = (N, n_0, S_p, S_s)$ be the prefix bi-tree associated with string $S$. The *compacted prefix bi-tree* of $S$ is a structure $C^P = (N^c, n_0, S^c_p, S^c_s)$, where $T^c = (N^c, n_0, S^c_p)$ is the compacted prefix-tree of $S$ and $S^c_s$, a function from $N^c \times \Sigma$ to $(N^c - \{n_0\}) \cup \{\text{NIL}\} \cup \{*\}$ is defined, for all $n' \in N^c$ and $\sigma$ in $\Sigma$, by

$$S^c_s(n', \sigma) = \begin{cases}\text{NIL} & \text{if } S_s(n', \sigma) = \text{NIL}\\ S_s(n', \sigma) & \text{if } S_s(n', \sigma) \in N^c\\ * & \text{otherwise.}\end{cases}$$

In order to derive a useful characterization of those non-NIL s-branches of $B^P$ which also occur in $C^P$, we present the following lemma.

*Lemma 4:*

Let $n \neq n_0$ be a node of $B^P = (N, n_0, S_p, S_s)$ with degree $d_n$ (in $T_p$) and let $n' = F_s(n)$ have degree $d_{n'}$. Then $d_{n'} \geqq d_n$.

*Proof:*

If n is a leaf of $T_p$, $d_n = 0$ and surely $d_{n'} \geqq d_n$. Consider, then, a son of n, say $\hat{n} = S_p(n, \hat{\sigma})$, and let $\hat{n}' = F_s(\hat{n})$ with $\hat{n} = S_s(\hat{n}', \hat{\sigma})$. From the definition of a bi-tree, it follows that $\omega = W(n) = \sigma\omega'$, where $\omega' = W(n')$. But $W(\hat{n}) = \sigma\omega'\hat{\sigma}$, so $W(\hat{n}') = \omega'\hat{\sigma}$. It follows that $S(n', \hat{\sigma}) = \hat{n}'$. Thus, for every son of n there exists a son of n', and the lemma is established.

QED

Our desired characterization of $S^c_s$ can now be established.

*Theorem 3:*

Let $C^P = (N^c, n_0, S^c_p, S^c_s)$ be the compacted prefix bi-tree of the string $S$. If $n \neq n_0$ is a node of $C^P$, then there exists a node $n' \in N^c$ and a $\sigma \in \Sigma$ such that $S^c_s(n', \sigma) = n$.

*Proof:*

Let $B^P = (N, n_0, S_p, S_s)$ be the prefix bi-tree of $S$, and let $n' = F_s(n)$. If $d_n \geqq 2$, then $d_{n'} \geqq 2$. Let $\tilde{n} = F(n)$ and $\tilde{n}' = F_s(\tilde{n}) = F_s(\tilde{n}) = F(n')$. If $d_{\tilde{n}} \geqq 2$ then $d_{\tilde{n}'} \geqq 2$. But n is a node in $N^c$ and either $d_{\tilde{n}} \geqq 2$ or $d_{\tilde{n}} \geqq 2$. It follows that node $n' = F_s(n)$ is also a node in $N^c$. Clearly, there also exists a $\sigma \in \Sigma$ such that $S^c_s(n', \sigma) = n$.

QED

What we have just shown is that every $n \neq n_0$ in $N^c$ has both a p-father and an s-father. A consequence of this fact is that $T^c_s = (N^c, n_0, S^c_s)$ is also a t-ary tree when both NIL and * values of $S^c_s$ are taken to be pointers to empty sub-trees.

The difference between NIL values of $S^c_s$ and * values of $S^c_s$ is important to our algorithm for calculating $C^P$. This procedure differs from Algorithm B in two important respects. First, when a type 2c construction is required, the father of $n_i$ and $\hat{n}$, $\ddot{n}$, is connected directly to $\tilde{n}$. Second, when a type 1 construction is required, but either $\ddot{n}$ or $\hat{n}$ is not already present, an insertion is made between two nodes in the tree.

As before, let $C_i^p$ be the compacted prefix bi-tree of $S_i$. Algorithm C computes $C_i^p$ from $C_{i+1}^p$. (Note that $LG_n$ gives the length of the walk $lg(W(n))$ in the non-compacted prefix bi-tree.)

<u>Algorithm C</u> (Construction of $C_i^p$ from $C_{i+1}^p$)

C1. [Initialize] Set $\ddot{n}' \leftarrow n_{i+1}$ and set j to the index of $S\langle i\rangle$. ($W(n_{i+1}) = I_{i+1}(i+1)$ and $S\langle i\rangle = \sigma_j$.)

C2. [Check label] If $L_{\ddot{n}'}(j)$ = NIL go to step C3; otherwise go to step C4. (When C4 is entered, $\ddot{\omega}' = W(\ddot{n}')$.)

C3. [Label and walk] Set $L_{\ddot{n}'}(j) \leftarrow i+1$ and $n_i' \leftarrow \ddot{n}'$. If $n_i' \neq n_0$, set $\ddot{n}' \leftarrow F^c(n_i')$ and go to step C2. Otherwise, set $\ddot{n} \leftarrow n_0$ and go to step C14.

C4. [Find $\hat{n}'$] Set $\hat{n}' \leftarrow S^c(\ddot{n}', S\langle L_{\ddot{n}'}(j)+LG_{\ddot{n}'}\rangle)$.

C5. [Is $\hat{n}$ in $B_{i+1}^p$?] If $S_s^c(\hat{n}',\sigma_j) \neq$ NIL, go to step C10. (This is the case when $\hat{\omega}$ is not a prefix identifier of $S_{i+1}$.)

C6. [Is $\ddot{n}$ in $C_{i+1}^p$?] If $S_s^c(\ddot{n}',\sigma_j) \neq$ NIL, go to step C9; otherwise, set $\tilde{n}' \leftarrow F^c(\ddot{n}')$.

C7. [Find $\tilde{n}$] If $S_s^c(\tilde{n}',\sigma_j) \neq$ NIL, set $\tilde{n} \leftarrow S_s^c(\tilde{n}',\sigma_j)$ and go to step C8; otherwise, set $S_s^c(\tilde{n}',\sigma_j) \leftarrow *$, $\tilde{n}' \leftarrow F^c(\tilde{n}')$, and repeat step C7.

C8. [Add $\ddot{n}$] Add node $\ddot{n}$ by setting $S^c(\tilde{n}, S\langle J_{\tilde{n}}+LG_{\tilde{n}}\rangle) \leftarrow \ddot{n}$, $F^c(\ddot{n}) \leftarrow \tilde{n}$, and $S_s^c(\ddot{n}',\sigma_j) \leftarrow \ddot{n}$. Label $\ddot{n}$ by setting $J_{\ddot{n}} \leftarrow J_{\tilde{n}}$, $LG_{\ddot{n}} \leftarrow LG_{\ddot{n}'}+1$, and $L_{\ddot{n}}(k) \leftarrow L_{\tilde{n}}(k)$ for $1 \leqq k \leqq t$. (Note that $LG_{\ddot{n}} \geqq LG_{\tilde{n}}+1$.)

C9. [Add $\hat{n}$] Add node $\hat{n}$ by setting $S^c(\ddot{n}, S\langle J_{\ddot{n}}+LG_{\ddot{n}}\rangle) \leftarrow \hat{n}$, $F^c(\hat{n}) \leftarrow \ddot{n}$, and $S_s^c(\hat{n}',\sigma_j) \leftarrow \hat{n}$. Label $\hat{n}$ by setting $J_{\hat{n}} \leftarrow J_{\ddot{n}}$, $LG_{\hat{n}} \leftarrow LG_{\hat{n}'}+1$, and $L_{\hat{n}}(k) \leftarrow L_{\ddot{n}}(k)$ for $1 \leqq k \leqq t$. Go to step C14.

C10. [Is insertion needed?] If $S_s^c(\hat{n}',\sigma_j) \neq *$ and $S_s^c(\ddot{n}',\sigma_j) \neq *$ go to step C14; otherwise, set $n_f' \leftarrow \ddot{n}'$.

C11. [Find father] If $S_s^c(n_f',\sigma_j) \neq *$, set $n_f \leftarrow S_s^c(n_f',\sigma_j)$ and go to step C12. Else, set $n_f' \leftarrow F^c(n_f')$ and repeat step C11. (This step must terminate, by Lemma 3!)

C12. [Insert $\hat{n}$, if required] If $S_s^c(\hat{n}',\sigma_j) \neq *$, go to step C13. Otherwise, set $n_s \leftarrow S^c(n_f, S\langle J_{n_f}+LG_{n_f}\rangle)$ and insert $\hat{n}$ between $n_f$ and $n_s$ by setting $S^c(n_f, S\langle J_{n_f}+LG_{n_f}\rangle) \leftarrow \hat{n}$, $F^c(\hat{n}) \leftarrow n_f$, $S^c(\hat{n}, S\langle J_{n_f}+LG_{n_f}+1\rangle) \leftarrow n_s$, $F^c(n_s) \leftarrow \hat{n}$, and $S_s^c(\hat{n}',\sigma_j) \leftarrow \hat{n}$. Label $\hat{n}$ by setting $J_{\hat{n}} \leftarrow J_{n_f}$, $LG_{\hat{n}} \leftarrow LG_{\hat{n}'}+1$, and $L_{\hat{n}}(k) \leftarrow L_{n_f}(k)$ for $1 \leqq k \leqq t$.

C13. [Insert $\ddot{n}$, if required] If $S_s^c(\ddot{n}',\sigma_j) \neq *$, go to step C14. Otherwise, set $n_s \leftarrow S^c(n_f, S\langle J_{n_f}+LG_{n_f}\rangle)$ and insert n between $n_f$ and $n_s$ by setting $S^c(n_f, S\langle J_{n_f}+LG_{n_f}\rangle) \leftarrow \ddot{n}$, $F^c(\ddot{n}) \leftarrow n_f$, $S^c(\ddot{n}, S\langle J_{n_f}+LG_{n_f}+1\rangle) \leftarrow \ddot{n}_s$, $F^c(n_s) \leftarrow \ddot{n}$, and $S_s^c(\ddot{n}',\sigma_j) \leftarrow n$. Label $\ddot{n}$ by setting $J_{\ddot{n}} \leftarrow J_{n_f}$, $LG_{\ddot{n}} \leftarrow LG_{\ddot{n}'}+1$, and $L_{\ddot{n}}(k) \leftarrow L_{n_f}(k)$ for $1 \leqq k \leqq t$.

C14. [Add $n_i$] Add node $n_i$ by setting $S^c(\ddot{n}, S\langle i+LG_{\ddot{n}}\rangle) \leftarrow n_i$, $F^c(n_i) \leftarrow \ddot{n}$, and $S_s^c(n_i',\sigma_j) \leftarrow n_i$. Label $n_i$ by setting $J_{n_i} \leftarrow i$, $LG_{n_i} \leftarrow LG_{n_i'}+1$, and $L_{n_i}(k) \leftarrow$ NIL for $1 \leqq k \leqq t$. Stop.

*Remark:*

Several of the steps in Algorithm C could well be combined into a parametized procedure.

To obtain the compacted prefix bi-tree $C^p$ for a string $S$ of length m, successively obtain $C_m^p, C_{m-1}^p, \ldots, C_1^p = C^p$. The run time of this procedure is of $O(m)$, but in order to demonstrate this, we need to develop a few new ideas; step C11 of Algorithm C may be executed several times, and no labels are added, so the analysis used for Algorithm B does not apply.

Let $C_i^p = (N_i^c, n_0, S_p^c, S_s^c)$ be the compacted prefix bi-tree of $S_i$ with node $n_i$ having $W(n_i) = I_i(i)$. Define the *Height* of node n in $C_i^p$, $h_i(n)$ to be the number of distinct ancestors of n in $C_i^p$. We wish to show that the number of steps executed in Algorithm C in constructing $C_i^p$ from $C_{i+1}^p$ is of $O(\delta_i)$, where $\delta_i = h_{i+1}(n_{i+1})-h_i(n_i)+1$. (The constant insures that $\delta_i \geqq 0$.) First, observe that Theorem 3 insures that every ancestor of $n_i$ except for $n_0$ has an s-father in $C_{i+1}^p$ which is an ancestor of $n_{i+1}$. This implies that $h_i(n_i) \leqq h_{i+1}(n_i')+1$.

Next, observe that all steps in Algorithm C are executed exactly once, except possibly steps C2, C3, C7, and C11. Steps C2 and C3 are executed $h_{i+1}(n_{i+1})-h_{i+1}(\ddot{n}')$ times. In the case of a type 1 construction which requires insertion, step C11 is executed $h_{i+1}(\ddot{n}')-h_{i+1}(n_f')$ times; in the case of a type 2b or 2c construction, step C7 is executed $h_{i+1}(\ddot{n}')-h_{i+1}(\tilde{n}')$ times. Thus the total number of C2, C3, C7, and C11 steps in the case of a type 1 construction without insertion or a type 2a construction is $h_{i+1}(n_{i+1})-h_{i+1}(\ddot{n}')$; for the case of a type 1 construction with insertion the total number is $h_{i+1}(n_{i+1})-h_{i+1}(n_f')$, and when a type 2b or type 2c construction is required, $h_{i+1}(n_{i+1})-h_{i+1}(\tilde{n}')$. But $h_i(n_i) \leqq h_{i+1}(n)+3$, where n is $\ddot{n}'$, $\tilde{n}'$, or $n_f'$. Thus, in all cases the total number of steps in Algorithm C is of order $h_{i+1}(n_{i+1})-h_i(n_i)+1$.

To prove that $C^p$ can be found in time $O(m)$, observe $\sum_{i=1}^{m-1} \delta_i = m+h_m(n_m)-h_1(n_1)$. Since $h_i(n_i) \leqq m-i+1$, $\sum_{i=1}^{m-1} \delta_i$ is of $O(m)$.

## IV. Applications

In this section, we indicate how to use compacted prefix-trees to solve various pattern matching problems in linear time.

*Problem 1:* (Basic Pattern Matching Problem)

Given a string $S$ of length $\ell_1$, and pattern $P$ of length $\ell_2$, find all positions i of $S$ such that $P = S\langle i{:}i+\ell_2-1\rangle$.

*Solution:*

Append a right end marker to $S$ and construct $T_p^c$ using Algorithm C. Determine whether any prefix of $P$ is a prefix of some prefix identifier of $S$, or vice versa by walking from the root of $T_p^c$ following branches labelled with symbols from $P$. If, at some stage of the walk, a node n is reached at level $LG_n$ and labelled with $J_n$, check the value of $S^c(n,P\langle 1+LG_n\rangle)$. If this value is NIL and if n is not a leaf, then $P$ does not match $S$ anywhere. If n is a leaf, then $P$ can possibly match $S$ at only one position, namely $J_n$. To see whether the match is valid, check for identity of $P\langle 1+LG_n+k\rangle$ and $S\langle J_n+LG_n+k\rangle$ for $k = 0,\ldots,\ell_2-LG_n-1$. (If $J_n+\ell_2-1 \geq \ell_1$, no match exists.) Next, consider the case $S^c(n,P\langle 1+LG_n\rangle) = n'$. If $LG_{n'} = LG_n+1$, simply continue the walk. On the other hand, if $LG_{n'} = LG_n+q$, and $q > 1$, then it is necessary to compare $P\langle 1+LG_n+k\rangle$ and $S\langle J_n+LG_n+k\rangle$ for $k = 1,\ldots,q-1$. Lack of equality for any k indicates no match; equality for all k allows the walk to be continued. Finally, consider the case where a $LG_n = \ell_2$. In this event, each leaf in the sub-tree rooted at n is labelled with the position of a match within $S$. A simple tree walk of this sub-tree finds these positions.

*Problem 2:* (Pattern match of several patterns with one string)

Given a string $S$ of length $\ell'$ and patterns $P_1,P_2,\ldots,P_q$ of lengths $\ell_1,\ell_2,\ldots,\ell_q$, find all matches of each pattern in $S$.

*Solution:*

Simply walk each pattern individually through $T_p^c$ as in the solution to Problem 1. Note that the total effort is of $O(\ell'+\ell_1+\ldots\ell_q)$. Note also that the Knuth-Pratt-Morris technique does not fare as well for this problem, since every symbol of $S$ is examined $\ell'$ times. However, Karp [8] has extended their technique and has developed an alternate linear time solution to Problem 2.

*Problem 3:* (Internal Matching)

Given a string $S$ of length $\ell$, find for each position i in $S$ another position P(i) in $S$ such that the longest common prefix of $S_i$ and $S_{P(i)}$ of length M(i) is no shorter than the longest common prefix of $S_i$ and $S_j$, $j \neq i$ and $j \neq P(i)$.

*Solution:*

Append an endmarker to $S$ and construct $T_p^c$. Locate (in constant time) the leaf n labeled $J_n = i$. If n' is a brother of n, then $P(i) = J_{n'}$ and the maximal match is of length $M(i) = LG_n-1$.

*Problem 4:* (External Matching)

Given two strings $\overline{S}$ and $\hat{S}$, of lengths $\overline{\ell}$ and $\hat{\ell}$, find for every position i in $\overline{S}$ the position P(i) and length M(i) of the longest match $\overline{S}\langle i{:}i+M(i)-1\rangle = \hat{S}\langle P(i){:}P(i)+M(i)-1\rangle$.

*Solution:*

Form the string $S = \overline{S}{:}\hat{S}\vdash$ where ":" is a distinct separator symbol, and construct the compacted prefix-tree $T_p^c$ of $S$. We proceed as in the solution to problem 3, except some care must be taken not to find matches entirely within $\overline{S}$. Assume that M(j) and P(j) have been obtained for $i+1 \leq j \leq \overline{\ell}$. To obtain M(i) and P(i), walk from the leaf n labelled with $J_n = i$ towards the root until a node n' is found with $J_{n'} > i$. (From the definition of J, $J_{n'} \geq J_n$ when n' is an ancestor of n.) If $J_{n'} > \overline{\ell}$, then $P(i) = J_{n'}$ and $M(i) = LG_{n'}$; otherwise, $P(i) = P(J_{n'})$ and $M(i) = M(J_{n'})$. Note that the total number of steps in finding P(i) and M(i) for $1 \leq i \leq \overline{\ell}$ is of $O(\overline{\ell}+\hat{\ell})$ since each node in $T_p^c$ is examined at most two times. This construction of the *match function* M and the *position function* P has direct application to the File Transmission Problem, as discussed in [9]. Note also that Problems 1 and 2 can be solved with variants of this solution.

We leave it to the reader to work out the variants of our methods required to solve Problems 1 and 2 of [4] for strings.

*Discussion:*

The techniques of this paper do not appear powerful enough to solve directly some interesting related pattern matching problems. For example, when "don't-care" elements are introduced, the best known results [5] suggest that an n log n algorithm may be possible, but none has yet been found. Also, the "sub-sequence" problem, mentioned in [6], has, at present, only an $n^2$ solution.

## Acknowledgments

This work was stimulated by extensive discussion with Robert W. Tuttle of Yale. Discussion with Albert R. Meyer, Michael J. Fischer, and Vaughan R. Pratt, all of MIT, helped develop the exposition. Also, a careful reading of the manuscript by Andrew H. Sherman of Yale led to several changes and improvements.

## References

1. Morris, James H., and Vaughan R. Pratt, "A linear pattern-matching algorithm," TR-40, Computing Center, University of California at Berkeley, 1970.
2. Knuth, Donald E., and Vaughan R. Pratt, "Automata theory can be useful," unpublished manuscript.
3. Cook, S. A., "Linear time simulation of deterministic two-way pushdown automata," Proceedings of IFIP Congress 71 (PA-2), North-Holland Publishing Co., The Netherlands, 1971, 174-179.
4. Karp, Richard M., Raymond E. Miller, and Arnold L. Rosenberg, "Rapid identification of repeated patterns in strings, trees and arrays," Fourth Symposium on Theory of Computing, May 1972, 125-136.
5. Paterson, M. S., "String-matching and other products," presented at a congress sponsored by the Istituto per le Applicazioni del Calcolo del

Consiglio Nazionale delle Richerche, Rome, Italy, 1973, 14 pages.
6. Wagner, Robert A., and Michael J. Fischer, "The string to string correction problem," unpublished manuscript, 13 pages.
7. Knuth, Donald E., The Art of Computer Programming, Volume 1, Fundamental Algorithms, Addison-Wesley, Reading, Massachusetts, 1968, 305-422.
8. Karp, Richard M., personal communication.
9. Weiner, Peter, and Robert W. Tuttle, "The file transmission problem," to be presented at the National Computer Conference, New York City, June 1973. Yale Computer Science Research Report #16.

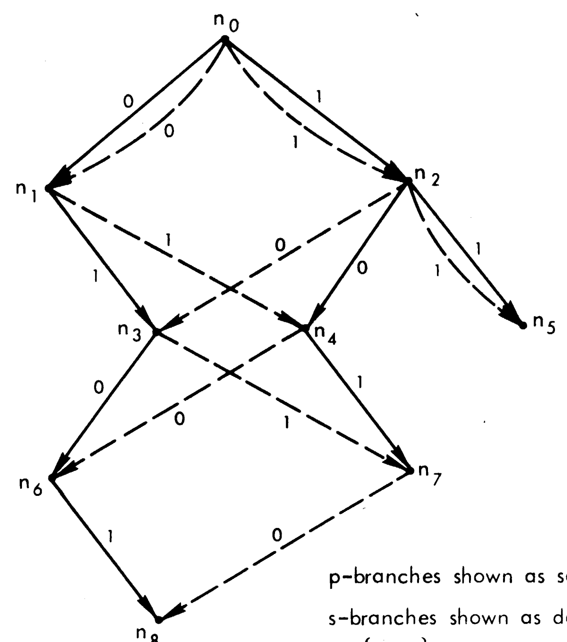


Figure 1. A bi-tree with 9 nodes.

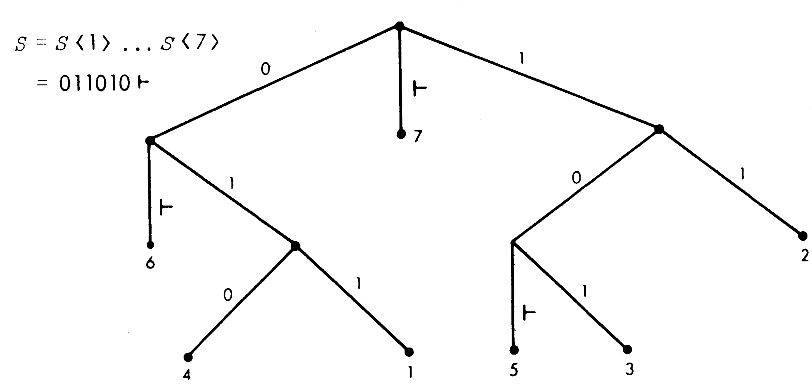


Figure 2. Prefix-tree of $S$.

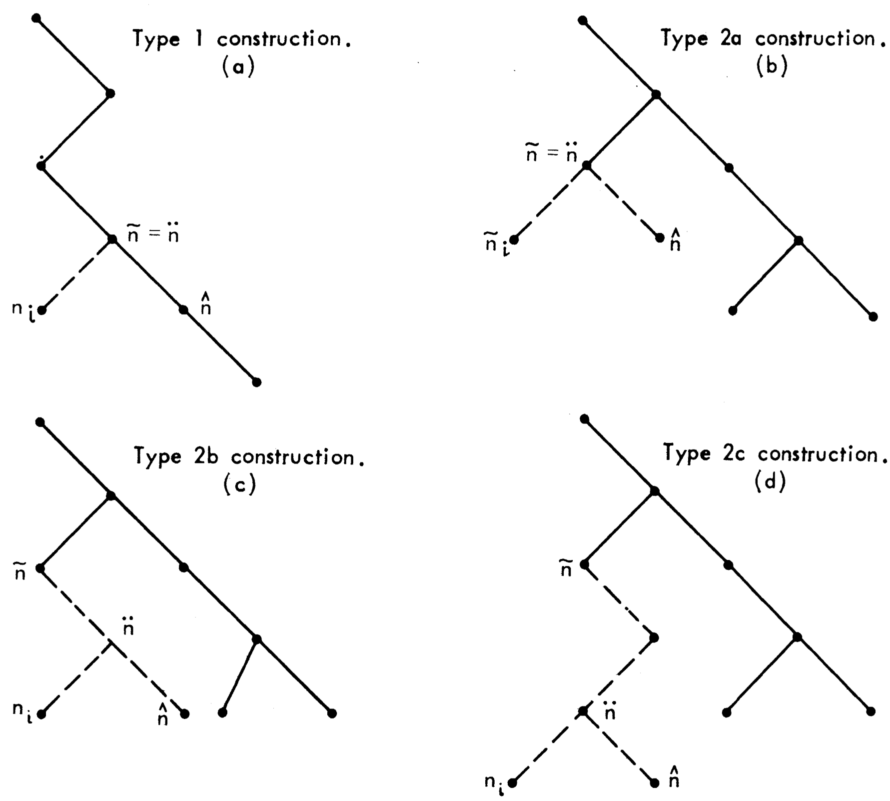


Figure 3.

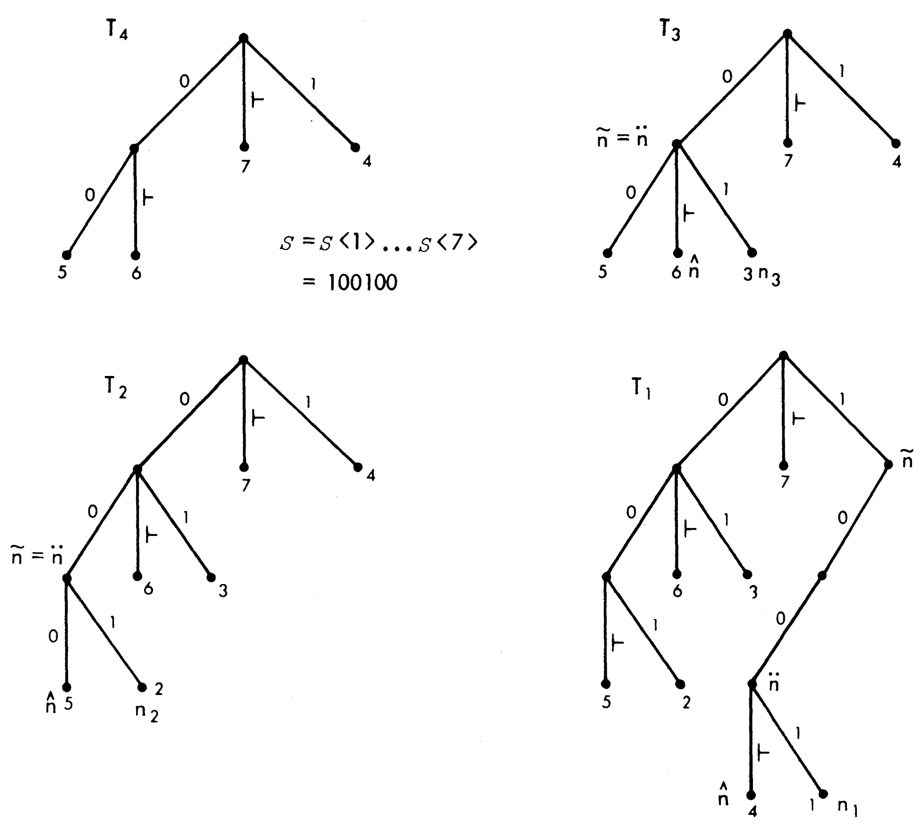


Figure 4.

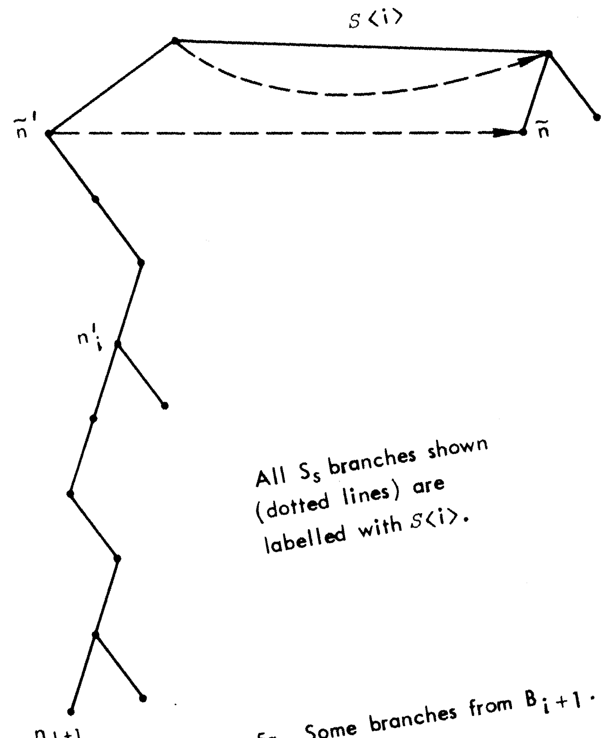


Figure 5a. Some branches from $B_{i+1}$.

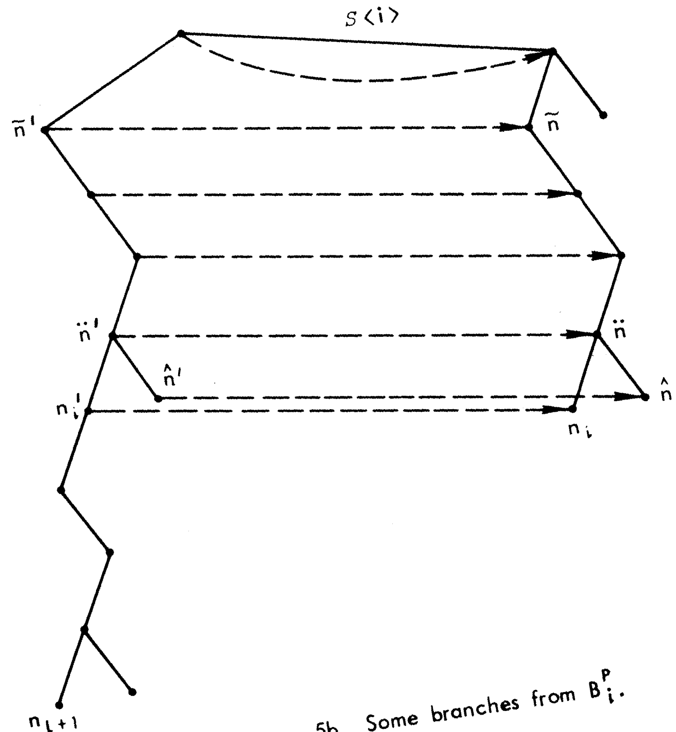


Figure 5b. Some branches from $B^P_i$.

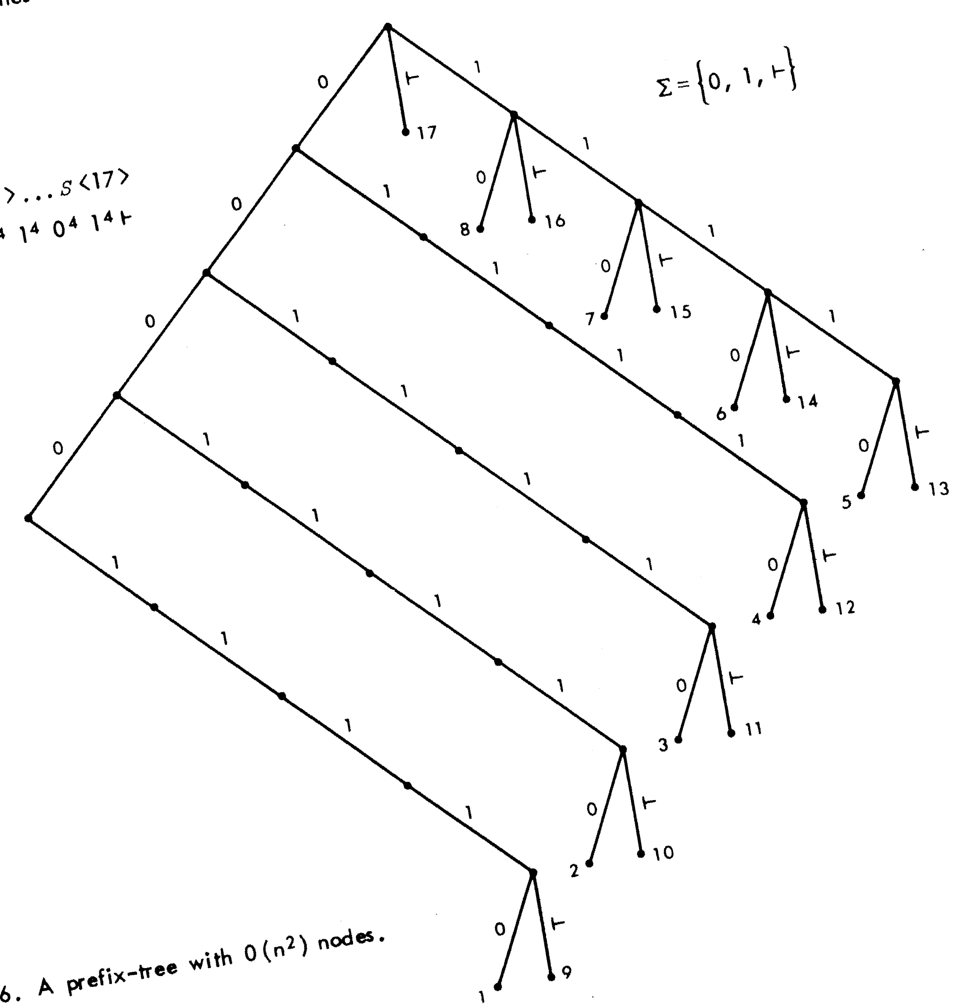


Figure 6. A prefix-tree with $O(n^2)$ nodes.