On the Tree Inclusion Problem

Yangjun Chen[1], Yibin Chen[2]

Dept. Applied Computer Science, University of Winnipeg, Canada
[1]y.chen@uwinnipeg.ca, [2]chenyibin@gmail.com

Abstract - The ordered tree inclusion is an interesting problem, by which we will check whether a pattern tree P can be included in a target tree T, in which the order of siblings is significant. In this paper, we propose an efficient algorithm for this problem.Its time complexity is almost linear in the size of T and P. Up to now the best algorithm for this problem needs quadratic time.

Keywords: tree inclusion; ordered labeled trees; tree matching.

1. Introduction

Let T be a rooted tree. We say that T is *ordered* and *labeled* if each node is assigned a symbol from an alphabet Σ and a left-to-right order among siblings in T is specified.

A tree T consisting of a specially designated node $root(T) = t$ (called the root of the tree) and a forest $<T_1, ..., T_k>$ (where $k \geq 0$) is denoted as $<t; T_1, ..., T_k>$. We also call T_j $(1 \leq j \leq k)$ a direct subtree of t.

The preorder of a forest $F= <T_1, ...,T_k>$ is the order of the nodes visited during a preorder traversal. A preorder traversal of a forest $<T_1, ...,T_k>$ is as follows. Traverse the trees $T_1, ...,T_k$ in ascending order of the indices in preorder. To traverse a tree in preorder, first visit the root and then traverse the forest of its subtrees in preorder. The postorder is defined similarly, except that in a postorder traversal the root is visited after traversing the forest of its subtrees in postorder. We denote the preorder and postorder numbers of a node v by $pre(v)$ and $post(v)$, respectively.

Let u, v be two nodes in T.If there is path from node u to node v, we say, u is an ancestor of v and v is a descendant of u. In this paper, by *ancestor* (*descendant*), we mean a proper ancestor (descendant), i.e., $u \neq v$. Using the preorder and postorder numbers, the ancestorship can be easily checked:

$u$is an ancestor of v if and only if $pre(u) <pre(v)$ and $post(v) <post(u)$. (See Exercise 2.3.2-20 in [5], page 347.)

Similarly,$u$is said to be to the left of v if they are not related by the ancestor-descendant relationship and v follows $u$when we traverse Tin preorder. Then, $u$is to the left of v if and only if $pre(u) <pre(v)$ and $post(u) <post(v)$.

In the following, we use $\prec$ to represent the left-to-right ordering. Also, $v \preceq v'$ iff$v \prec v'$ or $v = v'$. We will also use $V(T)$ and $E(T)$ to represent the set of nodes and the set of edges in F, respectively.

The following definition is due to [4].

Definition 1 Let F and G be labeled ordered forests. We define an ordered embedding (φ, G, F) as an injective function $\varphi: V(G) \rightarrow V(F)$ such that for all nodes $v, u \in V(G)$,

i) label(v) = label($\varphi(v)$); (label preservation condition)

ii) v is an ancestor of u iff$\varphi(v)$ is an ancestor of $\varphi(u)$, i.e., $pre(v) <pre(u)$ and $post(u) <post(v)$ iff$pre(\varphi(v))<pre(\varphi(u))$ and $post(\varphi(u))<post(\varphi(v))$; (ancestor condition)

iii) v is to the left of u iff$\varphi(v)$ is to the left of $\varphi(u)$, i.e., $pre(v) <pre(u)$ and $post(v) <post(u)$ iff$pre(\varphi(v))<pre(\varphi(u))$ and $post(\varphi(v)) <post(\varphi(u))$. (Sibling condition)

If there exists such an injective function from$V(G)$ to $V(F)$, we say, F includes G, F contains G, F covers G, or say, G can be embedded in F.

Fig. 1 shows an example of an ordered inclusion.

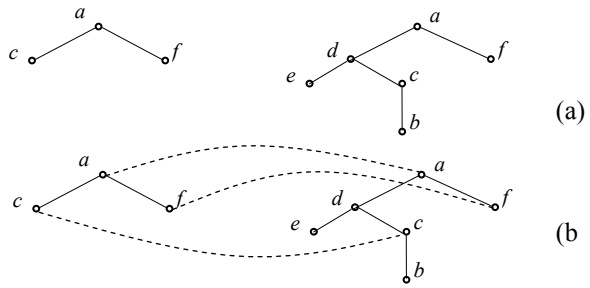

Figure 1: (a) The tree P on the left can be included in the tree T on the right; (b) an embedding of P in T.

Throughout the rest of the paper, we refer to the labeled ordered trees simply as trees.

The ordered tree inclusion problem was initially introduced by Knuth [5], where only a sufficient condition for this problem is given. Its first polynomial time algorithm was presented by Kilpeläinen and Mannila [4] with $O(|T|\cdot|P|)$ time and space being used. Most of the later improvements are refinements of this algorithm.

Recently, a break-through is achieved by Bille and Gørtz [1]. They got a space-economical algorithm with its space overhead bounded by$O(|T| + |P|)$, but with its time complexity bounded by

$$min\begin{cases} O(|T|\cdot|\text{leaves}(P)| \\ O(|\text{leaves}(T)|\cdot|\text{leaves}(P)|\cdot\text{loglog}|\text{leaves}(P)| + \\ O(|T|\cdot|P|/(\log|T|) + \end{cases}$$

whereD_T (resp. D_P) is the depth of T (resp. P), and leaves(T) (resp. leaves(P)) stands for the set of the leaves of T (resp. P).

In [3], a top-down algorithm was first proposed. Its space requirement is also bounded by $O(|T| + |P|)$. However, its time complexity is not polynomial, as shown in [6].

In this paper, we present a new algorithm for this problem. Its space overhead remains$O(|T|+|P|)$, but its time complexity is

reduced to $O(|T| \cdot \log D_P)$.

The tree inclusion problem on unordered trees is *NP*-complete [4] and not discussed in this paper.

2. Algorithm

Now we begin to describe our algorithm. First, we definesome notations in 2.1. Then, in 2.2 and 2.3, we describe our algorithm in great detail.

2.1 Basic notations

Let $T = \langle t; T_1, ..., T_k \rangle$ ($k \geq 0$) be a tree and $G = \langle P_1, ..., P_q \rangle$ ($q \geq 0$) be a forest. We will use p_v to represent the virtual parent of $P_1, ..., P_q$. Then, in G, every node v, excerpt p_v, has a parent, denoted as $parent(v)$.

Consider a node $v \in V(G) \cup \{p_v\}$ with children $v_1, ..., v_r$. We use a pair $\langle i, v \rangle$ ($i \leq r$) to represent an ordered forest containing the first $i$subtrees of v: $\langle G[v_1], ..., G[v_i] \rangle$.

We are interested in a special kind of subtrees, called *left corners*, defined below.

Definition 1 (*Left corners*) A forest $\langle i, v \rangle$ in G is called a left corner of G if $v = p_v$ or$v$ is a node on the left-most path in P_1.

Clearly, if $v = p_v$,$\langle i, v \rangle$ stands for a left corner of G, consisting of the first $i$subtrees in G: $P_1, ..., P_i$.

In the following, we will refer to a left corner of G simply as a left corner if no confusion will be caused according to the context.

In addition, we use $\rho(G)$ to represent the left-most leaf node of G. Then, $\langle i, \rho(G) \rangle$ (with any $i \geq 0$) or $\langle 0, v \rangle$ (with any v in G) stands for an empty left corner.

We also use $\delta(v)$ to represent a link from a node v to the left-most leaf node in $G[v]$, as illustrated in Fig. 2.

Let v' be a leaf node in G. $\delta(v')$ is defined to be a link to v' itself. So in Fig. 4, we have $\delta(v_1) = \delta(v_2) = \delta(v_3) = v_3$. Denote by $\delta^{-1}(v')$ a set of nodes $\boldsymbol{x}$ such that for each $v \in \boldsymbol{x}$ $\delta(v) = v'$. Then, in Fig. 2, We have $\delta^{-1}(v_3) = \{v_1, v_2, v_3\}$, $\delta^{-1}(v_4) = \{v_4\}$, and $\delta^{-1}(v_5) = \{v_5\}$.

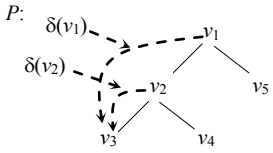

Figure 2. A pattern tree

Let p_1 be the root of P_1. We have $\rho(G) = \delta(p_1)$.

The outdegree of v in a tree is denoted by $d(v)$ while the height of v is denoted by $h(v)$, defined to be the number of edges on the longest downward path from v to a leaf. The height of a leaf node is set to be 0.

As with [3], we arrange two functions to check the tree inclusion. However, in [3], each function returns an integer j, indicating that the first j subtrees in G can be embedded in a target tree or a target forest while in our algorithm each function returns a left corner in G which can be embedded in the target.

If both the target and the pattern are forests, we call a function $A(\langle T_1, ..., T_k \rangle, \langle P_1, ..., P_q \rangle)$. If the target is a tree and the pattern is a forest, we call another function $B(T, \langle P_1, ..., P_q \rangle)$. But during the computation, they will be called from each other.

Let $\langle i, v \rangle$ be a left corner returned by $A(\langle T_1, ..., T_k \rangle, \langle P_1, ..., P_q \rangle)$ (or by $B(T, \langle P_1, ..., P_q \rangle)$). Then, the following properties are satisfied:

- If $i > 0$ and $v \neq \rho(G)$, it shows that
  - the first $i$subtrees of $v \in \delta^{-1}(\rho(G))$ can be embedded in$\langle T_1, ..., T_k \rangle$ (or in T).
  - for any $i' > i$, $\langle i', v \rangle$ cannot be embedded in $\langle T_1, ..., T_k \rangle$ (or in T); and
  - for any v's ancestor $u \in \delta^{-1}(\rho(G)) \cup \{p_v\}$, there exists no $j > 0$ such that $\langle j, u \rangle$ is able to be embedded in $\langle T_1, ..., T_k \rangle$ (or in T).
- If $i = 0$ or $v = \rho(G)$, it indicates that no left corner of G can be embedded in $\langle T_1, ..., T_k \rangle$ (or in T).

In this sense, we say, $\langle i, v \rangle$ is the *highest* and *widest* left corner which can be embedded in $\langle T_1, ..., T_k \rangle$ (or in T).
We notice that if $v = p_v$ and $i > 0$, it shows that $P_1, ..., P_i$ can be included in $\langle T_1, ..., T_k \rangle$ (or in T).

Finally, we say,a left corner $\langle i, v \rangle$ is higher than a node u(or another left corner $\langle j, u \rangle$)if$v$ is an ancestor of u.

2.2 *A*-function

First, let's have a look at a naïve way to evaluate$A(F, G)$, where $F = \langle T_1, ..., T_k \rangle$ and $G = \langle P_1, ..., P_q \rangle$.

1. Two index variables j, l are used to scan $T_1, ..., T_k$ and $P_1, ..., P_q$, respectively. Initially, j is set to 1, and l is set to 0. (They also indicate that $\langle P_1, ..., P_l \rangle$ has been successfully embedded in $\langle T_1, ..., T_j \rangle$.) In each step, we call $B(T_j, \langle P_{l+1}, ..., P_q \rangle)$.
2. Let $\langle i_j, v_j \rangle$be the return value of $B(T_j, \langle P_{l+1}, ..., P_q \rangle)$.If $v_j = parent(p_1)$, set $l$to be $l + i_j$. Otherwise, $l$is not changed. Set $j$to be $j + 1$. Go to (2).
3. The loop terminates when all T_j's or all P_l's are examined.

If $l > 0$ when the loop terminates, $B(T, G)$ returns $\langle l, parent(p_1) \rangle$, indicating that F contains $P_1, ..., P_l$.

Otherwise, $l = 0$, indicating that even P_1 alone cannot be embedded in any T_j ($j \in \{1, ..., k\}$). However, in this case, we need to continue to look for a highest and widest left corner $\langle i, v \rangle$ in G, which can be embedded in G. This is done as described below.

4. Let $\langle i_1, v_1 \rangle, ..., \langle i_k, v_k \rangle$ be the return values of $B(T_1, \langle P_1, ..., P_q \rangle), ..., B(T_k, \langle P_1, ..., P_q \rangle)$, respectively. Since $j = 0$, each $v_j \in \delta^{-1}(v')$ ($j = 1, ..., k$), where v' is the left-most leaf in P_1.
5. If each $i_j = 0$, return $\langle 0, \rho(G) \rangle$. Otherwise, there must be some v_j's such that $i_j > 0$. We call such a node a *non-zero point*. Find the first non-zero point v_f with children $w_1, ..., w_y$ such that v_f is not a descendant of any other non-zero point. Then, we will check $\langle T_{f+1}, ..., T_k \rangle$ against$\langle G[w_{i_{f+1}}], ..., G[w_y] \rangle$. Let x ($0 \leq x \leq y - i_f$) be a number such that $\langle G[w_{i_{f+1}}], ..., G[w_x] \rangle$ can be embedded

in $<T_{f+1}, ..., T_k>$. Thereturn value of $A(F, G)$ should be set to $<i_f + x, v_f>$.

In the above process, (1) - (3) are referred to as a *main checking* while (4) – (5) as a *supplement checking*.

We notice that in the supplement checking, only the first non-zero is utilized for forming the final result while all the other calls of the form$B(T_j, <P_1, ..., P_q>)$ (done in the main checking) are not usedat all. Their efforts for looking for the corresponding return values bring the void, and therefore should be avoided.

For this purpose, we introduce the concept of *cuts* to integrate a kind of control into the above working process.

Definition 2 A cut for a call of the form $A(F, G)$ is a node $u(\neq p_v) \in \delta^{-1}(\rho(G))$, indicating that if the supplement checking in $A(F, G)$ can only bring out a left corner $<i, v>$ not higher than u, the corresponding computation makes no contribution to the final result.

The following example helps for illustration.

Example 1 Consider target forest $F$and pattern forest G shown in Figure 3, in which each node in$F$ is identified with t_i, such as t_1, t_2, t_{11}, and so on; and each node in G is identified with p_j. Besides, each subtree rooted at t_i (resp. p_j) is represented by T_i (resp. P_j).

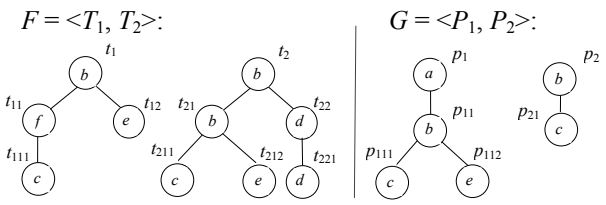

Figure 3. A target and a pattern forest

Initially, for$A(<T_1, T_2>, <P_1, P_2>)$, we will set its cut$u_0$ to be $\rho(G) = p_{111}$, imposing in fact no control on its supplement checking at all.When executing $B(T_1, <P_1, P_2>)$, its cut u_1 isset to be the same as u_0(i.e., $u_1 = u_0$).

It can be seen that T_1 is able to include only $G[p_{11}]$. So the return value of this call should be $<1, p_1>$. Then, for $B(T_2, <P_1, P_2>)$, the cut $u_2$should be set top_1, indicating that the supplement checking within $B(T_2, <P_1, P_2>)$ should be cut off if it can only produce a left corner not higher than p_1 since the result will not be used.

As will be seen later, during the execution of $B(T_2, <P_1, P_2>)$, $A(<T_{21}, T_{22}>, <P_1, P_2>)$ will be called and the cut u_{21} for this call is the same as$u_2 = p_1$. Then, after the main checking of$A(<T_{21}, T_{22}>, <P_1, P_2>)$, its supplement checking will be discarded since in its main checking the return values of$B(T_{21}, <P_1, P_2>)$ and $B(T_{21}, <P_1, P_2>)$ are$<1, p_1>$ and $<0, \rho(G)>$, respectively; and the supplement checking will not create a left corner higher p_1. □

With the cutsbeing considered, the A-function should be changed to take three inputs: $F=<T_1, ...,T_k>$, $G= <P_1, ..., P_q>$, and $u\in\delta^{-1}(p_1)$. (Initially, u is set to $\rho(G)$.) In the main checking of $A(F, G, u)$, the cut for each B-function call will be dynamically changed as described below.

i) At the very beginning, we will check whether u is higher thanp_1, where p_1 is the root of P_1. If it is the case, we simply return $<0, \rho(G)>$ since the computation will not make any contribution to the final result. Otherwise, we will do the following.

ii) For the first B-function call $B(T_1, <P_{l_1}, ..., P_q>, u_1)$ (where $l_1= 1$), set $u_1= u$. Let $<i_1, v_1>$ be its return value. We will call $B(T_2, , <P_{l_2}, ..., P_q>, u_2)$ in a next step. If $v_1 = parent(p_1)$, $l_2= i_1 + 1$) and u_2 is set to be p_{l_2}. If $v_1\neq parent(p_1)$, $l_2= l_1= 1$, and u_2 is set to be v_1.

iii) In general, let $<i_j, v_j>$ be the return value of $B(T_j, <P_{l_j}, ..., P_q>, u_j)$ for $j= 1, ..., x \le k$, $j_1 = 1$, $j_1 \le j_2 \le ... \le j_x \le q$. Let s be an integer such that $l_1 = ... = l_s = 1$, but$l_{s+1} > 1$. Then, for $2 \le j \le s$, we have

$$u_j = \begin{cases} v_{j-1}, & \text{if} v_{j-1} \text{ is higher than} u_{j-1} \text{ and } i_{j-1} > 0; \\ u_{i-1}, & \text{if} v_{j-1} \text{ is not higher than} u_{j-1} \text{ or } i_{j-1} = 0; \end{cases} \quad (2.1$$

and for $s + 1 \le j \le x$, we have

$$u_j = p_{l_j}. \quad (2.2)$$

The formula (2.1) shows how the cuts are changed before we find the first $T_s$which is able to embed some subtrees in G. After T_s, the cuts are determined in terms of the formula (2.2). Setting u_j to be p_{l_j} will effectively prohibit the supplement checking in the execution of $B(T_j, <P_{l_j}, ..., P_q>, u_j)$, which will definitely returna left corner not higher than $p_{j_l}$and therefore is useless.

After the main checking, the following checks will be conducted to determine whether a supplement checking will be carried out.

- If $l = q$, we will record the embedding.
- If$j < k$, we will continue to find a next embedding by making a recursive call $A(<T_{j+1}, ..., T_k>, <P_1, ..., P_q>, p_1)$.
- If there is at least an embedding, return $<q, p_v>$.
- If $0<l<q$, return $<l, p_v>$.
- If$l = 0$ and$f = 0$, return $<0, \delta(p_1)>$.
- Otherwise, a supplement checking will be conducted.

Let $<i_f, v_f>$be the return valueof some $B(T_j, <P_{l_j}, ..., P_q>, u_j)$ such that v_f is not a descendant of any other non-zero point. We will make a recursive call$A(<T_{f+1}, ..., T_k>, <G[w_{i_f+1}], ...,G[w_y]>, w_{i_f+1})$ for doing a supplement checking, where $w_1, ..., w_y$ are the children of v_f. We notice that the cut for this recursive call is set to be $w_{i_f+1}$to cut off a possible supplement checking in this execution.

In terms of the above discussion, we give the following algorithm.

function$A(F, G, u)$ (*Initially, $u = \rho(G)$.*)

input: $F = <T_1, ..., T_k>$, $G = <P_1, ..., P_q>$, u – a cut.

output: <i, v> specified above.

begin

1. **if** p_1 is a descendant of u **then** return <0, $\delta(p_1)$>;
2. j:= 1; l := 0; v := u; f := 0; i := 0;
3. **while** (l <q and $j \le k$) **do** (*main checking*)
4. { <i_j, v_j> := $B(T_j, <P_{l+1}, ..., P_q>, v)$
5. **if** ($v_j = p_v$) **then** {$l := l + i_j$; $v := p_{l+1}$;}
6. **else if** (v_j is an ancestor of v and $i_j > 0$)
   then {$v := v_j$; $i := i_j$; $f := j$;}
7. $j := j + 1$;
8. }
9. **if** $l = q$ **then** record the embedding;
10. **if** $j < k$ **then** {<i'', v''> := $A(<T_{j+1}, ..., T_k>, G, p_1)$;}
11. **if** there is at least an embedding **then** return <q, p_v>;
12. **if** $l > 0$ **then** return <l, p_v>;
13. **if** $f = 0$ **then** return <0, $\delta(p_1)$>;
14. let $w_1, ..., w_s$ be the children of v;
15. $j := f + 1$; (*supplement checking*)
16. <i', v'> := $A(<T_{j+1}, ..., T_k>, <G[w_{i+1}], ..., G[w_s]>, w_{i+1})$;
17. **if** ($v' = v$ and $i' > 0$) **then** return <$i + i'$, v>;

end

2.3 *B*-function

In $B(T, G, u)$, we need to distinguish between two cases.

Case 1: $G = <P_1>$; or

$G = <P_1, ..., P_q>$ ($q > 1$), but $|T| \le |P_1| + |P_2|$.

In this case, what we can do is to find whether P_1 or a left corner in P_1 can be embedded in $T = <t; T_1, ..., T_k>$. For this purpose, the following checkings should be conducted:

i) If t is a leaf node, we will check whether label(t) = label($\delta(p_1)$), where p_1 is the root of P_1. If it is the case, return <1, parent of $\delta(p_1)$>. Otherwise, return <0, $\delta(p_1)$>.

ii) If $|T| > 1$, but $|T| < |P_1|$ or $h(t) < h(p_1)$, we will make a recursive call $B(T, <P_{11}, ..., P_{1j}>, u)$, where $<P_{11}, ..., P_{1j}>$ is a forest of the subtrees of p_1. The return value of $B(T, <P_{11}, ..., P_{1j}>, u)$ is used as the return value of $B(T, G, u)$. It is because in this case, T is not able to include the whole P_1. So what we can do is to check T against $<P_{11}, ..., P_{1j}>$.

iii) If $|T| \ge |P_1|$ and $h(t) \ge h(p_1)$ (but $|T| \le |P_1| + |P_2|$), we further distinguish between two sub-cases:

- label(t) = label(p_1). In this case, we will call $A(<T_1, ..., T_k>, <P_{11}, ..., P_{1j}>, p_{11})$ or $A(<T_1, ..., T_k>, <P_{11}, ..., P_{1j}>, u)$, depending on whether $u = p_1$. If $u = p_1$, the cut for this call is set to be p_{11} because if the left corner returned by this call is not higher than p_{11}, it will not be used.
- label(t) ≠ label(p_1). In this case, we will call $A(<T_1, ..., T_k>, <P_1>)$.

In both cases, assume that the return value of $A(\ )$ is <i, v>. We need to do an extra checking:

- If label(t) = label(v) and $i = d(v)$, the return value of $B(T, G, u)$ is set to be <1, v's parent>.
- Otherwise, the return value of $B(T, G, u)$ is the same as <i, v>.

Case 2: $G = <P_1, ..., P_q>$ ($q > 1$), and $|T| > |P_1| + |P_2|$.

In this case, we will call $A(<T_1, ..., T_k>, G, u)$. Assume that the return value of $A(<T_1, ..., T_k>, G, u)$ is <i, v>. The following checkings will be continually conducted.

iv) If $v \ne p_1$'s parent, check whether label(t) = label(v) and $i = d(v)$. If it is not the case, the return value of $B(T, G, u)$ is the same as <i, v>. Otherwise, the return value of $B(T, G, u)$ will be set to <1, v's parent>.

v) If $v = p_1$'s parent, the return value of $B(T, G, u)$ is the same as <i, v>.

In terms of the above discussion, we give thefollowing formal description of the algorithm, in which $B'(\ )$ is used to handle Case 1 and $B''(\ )$ for case 2.

function $B(T, G, u)$ (*Initially, $u = \rho(G)$.*)
input: $T = <t; T_1, ..., T_k>$, $G = <P_1, ..., P_q>$, u – cut.
output: <i, v> specified above.

begin

1. **if** p_1 is a descendant of u **then** return <0, $\delta(p_1)$>;
2. **if** ($q = 1$ or $|T| \le |P_1| + |P_2|$) **then** return $B'(T, P_1, u)$
3. **else** return $B''(T, G, u)$;

end

function $B'(T, P, u)$ (*Case* 1*)

begin

1. let $T = <t; T_1, ..., T_k>$; let $P = <p; P_1, ..., P_j>$;
2. **if** t is a leaf **then** (*Case* 1 - (i)*)
3. { let δ(p) = v;
4. **if** label(t) = label(v) **then** return <1, v's parent>
5. **else** return <0, v>;
6. }
7. **if** ($|T| < |P| \vee h(t) < h(p)$) **then** return $B(T, <P_1, ..., P_j>, u)$;
   (*Case 1 - (ii)*)
8. **if** label(t) = label(p) (*Case* 1- (iii)*)
9. **then** {
   if p is a leaf **then** {v := p's parent; i := 1;}
10. **else** { **if** $u = p$
    then <i, v> := $A(<T_1, ..., T_k>, <P_1, ..., P_j>, p_1)$
11. **else** <i, v> := $A(<T_1, ..., T_k>, <P_1, ..., P_j>, u)$;
12. **if** label(t) = label(v) and $i = d(v)$
13. **then** {v := v's parent; i := 1; }
14. }
15. }
16. **else** <i, v> := $A(<T_1, ..., T_k>, <P>, u)$;
    (*If label(t) ≠ label(p), call $A(\ )$.*)
17. return <i, v>;
18. }

end

function $B''(T, G, u)$ (*Case* 2*)

begin

1. let $T = <t; T_1, ..., T_k>$;
2. <i, v> := $A(<T_1, ..., T_k>, G, u)$;
3. **if** $v \ne p_v$ (*Case* 2 - (iv)*)
5. **then** { **if** (label(t) = label(v)) $\wedge i = d(v)$
6. **then** return <1, v's parent>;

7. }
8. return<*i*, *v*>; (**Case* 2 - (v)*)
end

This algorithm is for Case 2. Again, the cut for the *B*-function isdirectly propagated to the subfunction calls of *A*( ) (see line 2.)

3. Conclusion

In this paper, a new algorithm is proposed to solve the ordered tree inclusion problem. Up to now, the best algorithm for this problem needs quadratic time. However, ours requires only $O(|T|\cdot\log D_P)$ time and $O(|T| + |P|)$ space, where T and P are a target and a pattern tree (forest), respectively; and D_P is the depth of P.

4. References

[1] P. Bille and I.L. Gørtz, The Tree Inclusion Problem: In Linear Space and Faster, *ACM Transaction on Algorithms*, Vol. 7, No. 3, Article 38, July 2011, pp. 38:1-38:47.

[2] W. Chen. More efficient algorithm for ordered tree inclusion. *Journal of Algorithm*s, 26:370-385, 1998.

[3] Y. Chen and Y.B. Chen, A New Tree Inclusion Algorithm, *Information Processing Letters* 98(2006) 253-262, Elsevier Science B.V.

[4] P. Kilpeläinen and H. Mannila. Ordered and unordered tree inclusion. *SIAM J. Comput*, 24:340-356, 1995.

[5] D.E. Knuth, *The Art of Computer Programming, Vol. 1 (1st edition)*, Addison-Wesley, Reading, MA, 1969.

[6] H.L Cheng and B.F Wang, On Chen and Chen's new tree inclusion algorithm, *Information Processing Letters*, 2007, Vol. 103, 14-18, Elsevier Science B.V.