Road map

- Introduction
- Data Model and HTML encoding
- Wrapper induction
- Automatic Wrapper Generation: Two Problems
- String Matching and Tree Matching
- **Multiple Alignments**
- Building DOM Trees
- Extraction Given a List Page: Flat Data Records
- Extraction Given a List Page: Nested Data Records
- Extraction Given Multiple Pages
- Summary

Multiple alignment

- Previous techniques were used to compare two strings or two trees.
- Pairwise alignment is not sufficient because a web page usually contain more than one data records.
- We need multiple alignment to compare more than two strings or two trees
- We discuss two techniques
  - Center Star method
  - Partial tree alignment.

Center star method

- This is a classic technique, and quite simple. It commonly used for multiple string alignments, but can be adopted for trees.
- Let the set of strings to be aligned be *S*. In the method, a string $s_c \in S$ that minimizes,

$$\sum_{s_i \in S} d(s_c, s_i) \qquad (3)$$

- is first selected as the **center string**. $d(s_c, s_i)$ is the distance of two strings.
- The algorithm then iteratively computes the alignment of rest of the strings with s_c.

The algorithm

```
CenterStar(S)
1.   choose the center star $s_c$ using Equation (3);
2.   initialize the multiple sequence alignment $M$ that contains only $s_c$;
4.   for each $s$ in $S$-$\{s_c\}$ do
5.       let $c^*$ be the aligned version of $s_c$ in $M$;
6.       let $s'$ and $c^{*\prime}$ be the optimally aligned strings of $s$ and $c^*$;
7.       add aligned strings $s'$ and $c^{*\prime}$ into the multiple alignment $M$;
8.       add spaces to each string in $M$, except, $s'$ and $c^{*\prime}$, at locations where new
             spaces are added to $c^*$
9.   endfor
10.  return multiple string alignment $M$
```

An example

Example 2: We have three strings, i.e., S = {ABC, XBC, XAB}. ABC is selected as the center string s_c. Let us align the other strings with ABC.

Iteration 1: Align c^* (= s_c) with s =XBC:

$$\begin{array}{lccc} c^{*\prime}: & \mathrm{A} & \mathrm{B} & \mathrm{C} \\ & & | & | \\ s': & \mathrm{X} & \mathrm{B} & \mathrm{C} \end{array}$$

$$\text{Update } M\text{:} \quad \begin{array}{ccc} \mathrm{A} & \mathrm{B} & \mathrm{C} \end{array} \rightarrow \begin{array}{ccc} \mathrm{A} & \mathrm{B} & \mathrm{C} \\ \mathrm{X} & \mathrm{B} & \mathrm{C} \end{array}$$

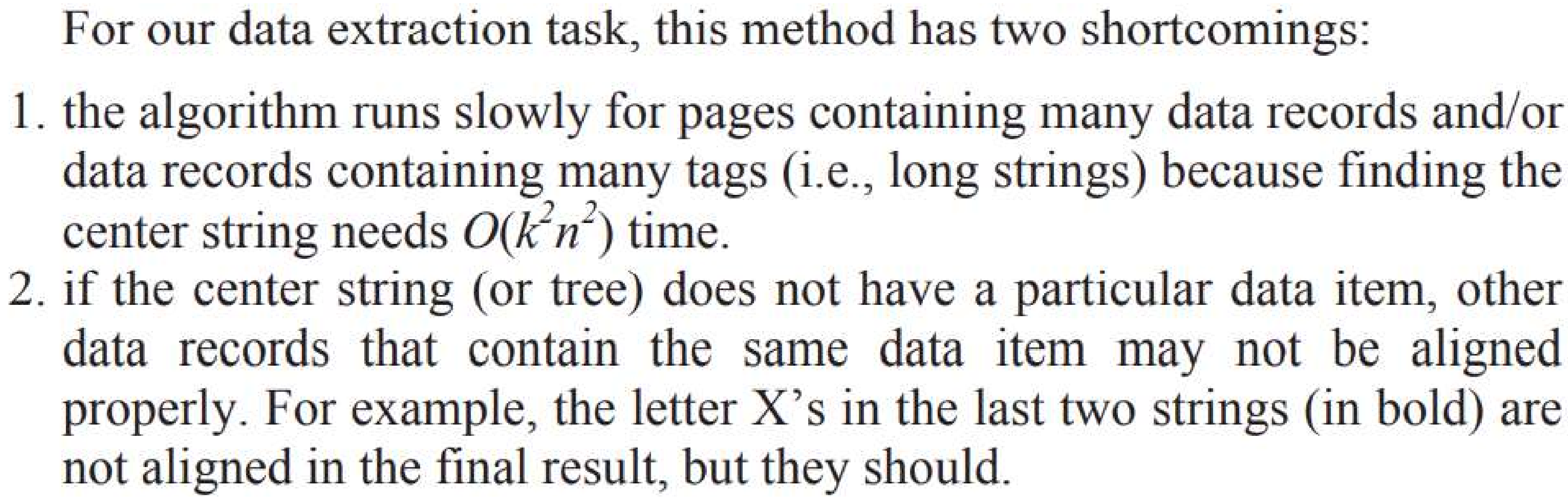
Iteration 2: Align c^* with s = XAB:

$$\begin{array}{lcccc} c^{*\prime}: & - & \mathrm{A} & \mathrm{B} & \mathrm{C} \\ & & | & | & \\ s': & \mathrm{X} & \mathrm{A} & \mathrm{B} & - \end{array}$$

$$\text{Update } M\text{:} \quad \begin{array}{ccc} \mathrm{A} & \mathrm{B} & \mathrm{C} \\ \mathrm{X} & \mathrm{B} & \mathrm{C} \end{array} \rightarrow \begin{array}{cccc} \boldsymbol{-} & \mathbf{A} & \mathbf{B} & \mathbf{C} \\ \boldsymbol{-} & \mathbf{X} & \mathbf{B} & \mathbf{C} \\ \mathbf{X} & \mathbf{A} & \mathbf{B} & \boldsymbol{-} \end{array}$$

The shortcomings

- Assume there are *k* strings in *S* and all strings have length *n*, finding the center takes $O(k^2n^2)$ time and the iterative pair-wise alignment takes $O(kn^2)$ time. Thus, the overall time complexity is $O(k^2n^2)$.



For our data extraction task, this method has two shortcomings:

1. the algorithm runs slowly for pages containing many data records and/or data records containing many tags (i.e., long strings) because finding the center string needs $O(k^2n^2)$ time.
2. if the center string (or tree) does not have a particular data item, other data records that contain the same data item may not be aligned properly. For example, the letter X's in the last two strings (in bold) are not aligned in the final result, but they should.

Shortcomings (cont …)

- Giving the cost of 1 for "changing a letter" in edit distance is problematic (e.g., A and X in the first and second strings in the final result) because of optional data items in data records.
- The problem can be partially dealt with by disallowing "changing a letter" (e.g., giving it a larger cost). However, this introduces another problem.
- For example, if we align only ABC and XBC, it is not clear which of the following alignment is better.

```
(1)  A  –  B  C          (2)  –  A  B  C
     –  X  B  C               X  –  B  C
```

Example 1

- Align the following strings
  A. ABRGQ
  B. BRTGZ
  C. AQRTQ
  D. AAGGQ
- Which of the strings is chosen as centre star?

Example 2

- Align the following strings
  - ERNSTY
  - QRSTXY
  - PRISTTY

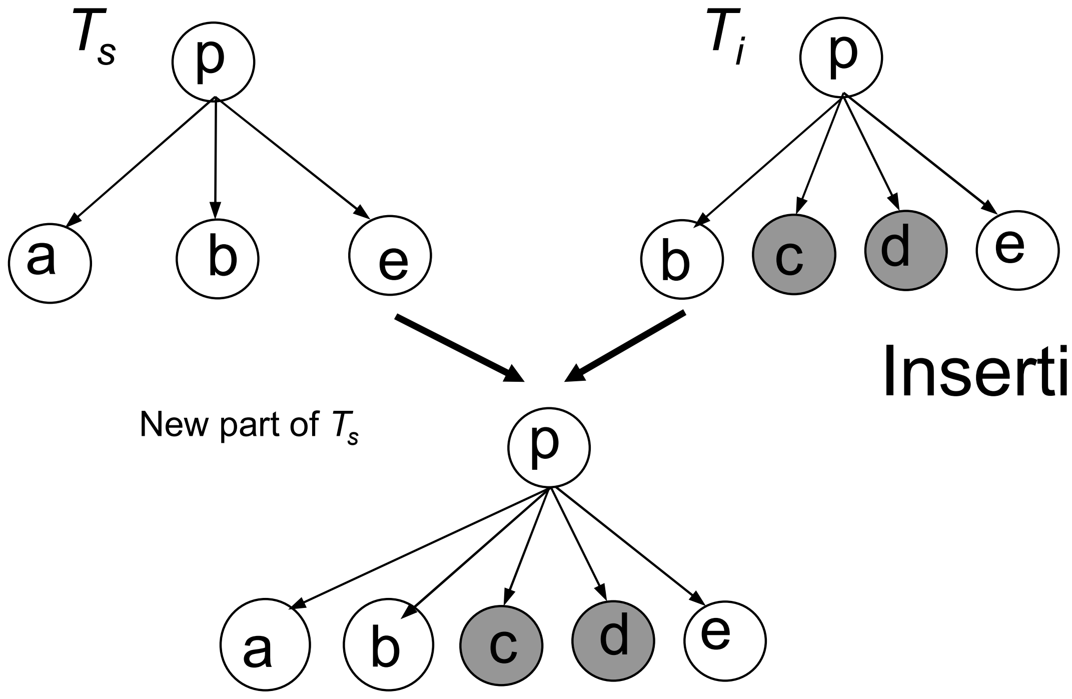
The partial tree alignment method

- Choose a seed tree: A seed tree, denoted by T_s, is picked with the maximum number of data items.
- The seed tree is similar to center string, but without the $O(k^2n^2)$ pair-wise tree matching to choose it.
- Tree matching:

  For each unmatched tree T_i ($i \neq s$),
  - match T_s and T_i.
  - Each pair of matched nodes are linked (aligned).
  - For each unmatched node n_j in T_i do
    - expand T_s by inserting n_j into T_s if a position for insertion can be uniquely determined in T_s.

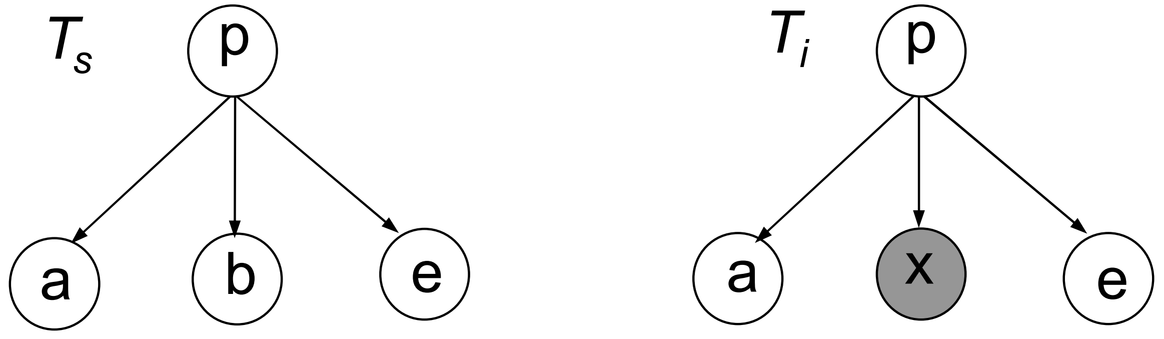
  The expanded seed tree T_s is then used in subsequent matching.

Partial tree alignment of two trees

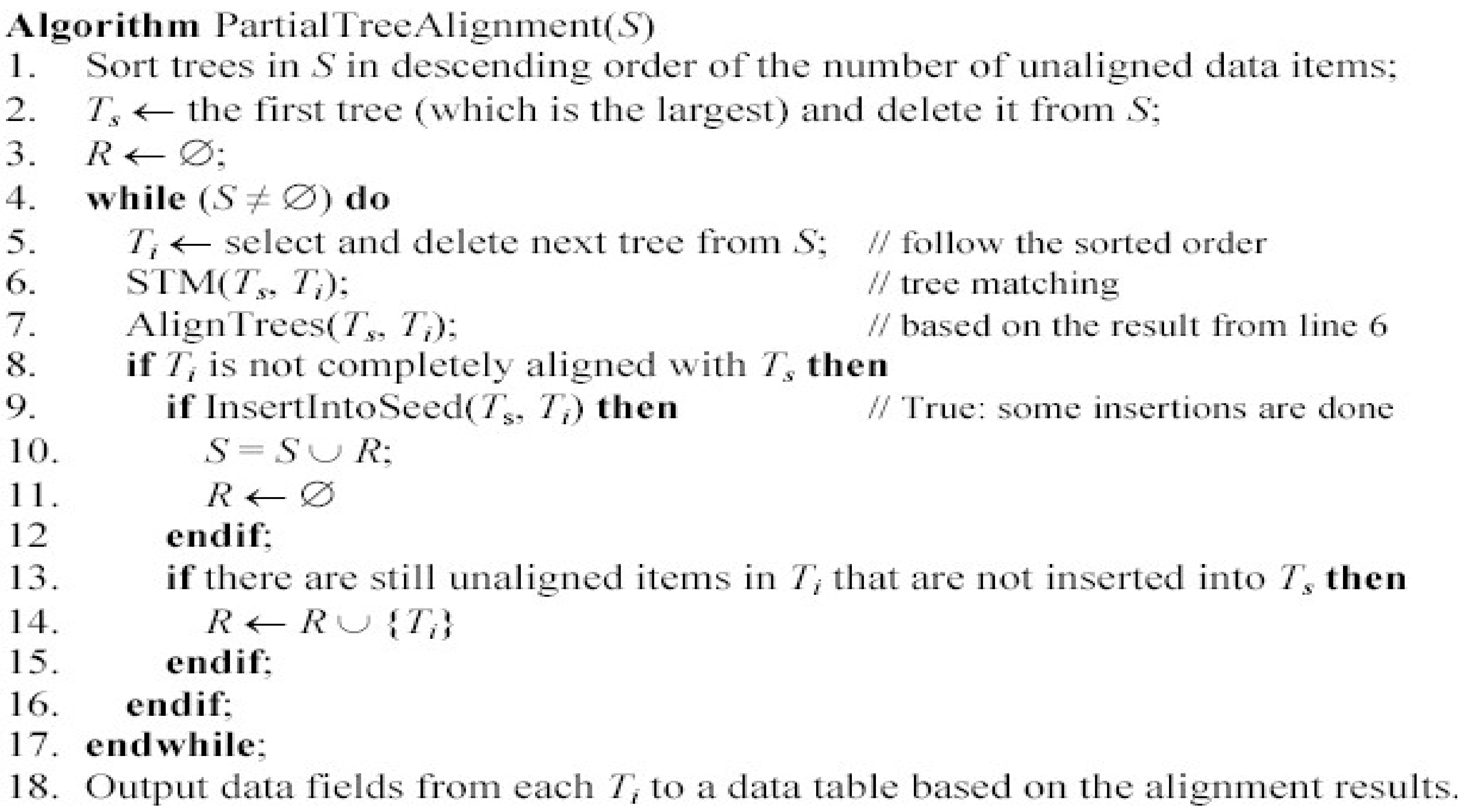


Insertion is possible

Insertion is not possible



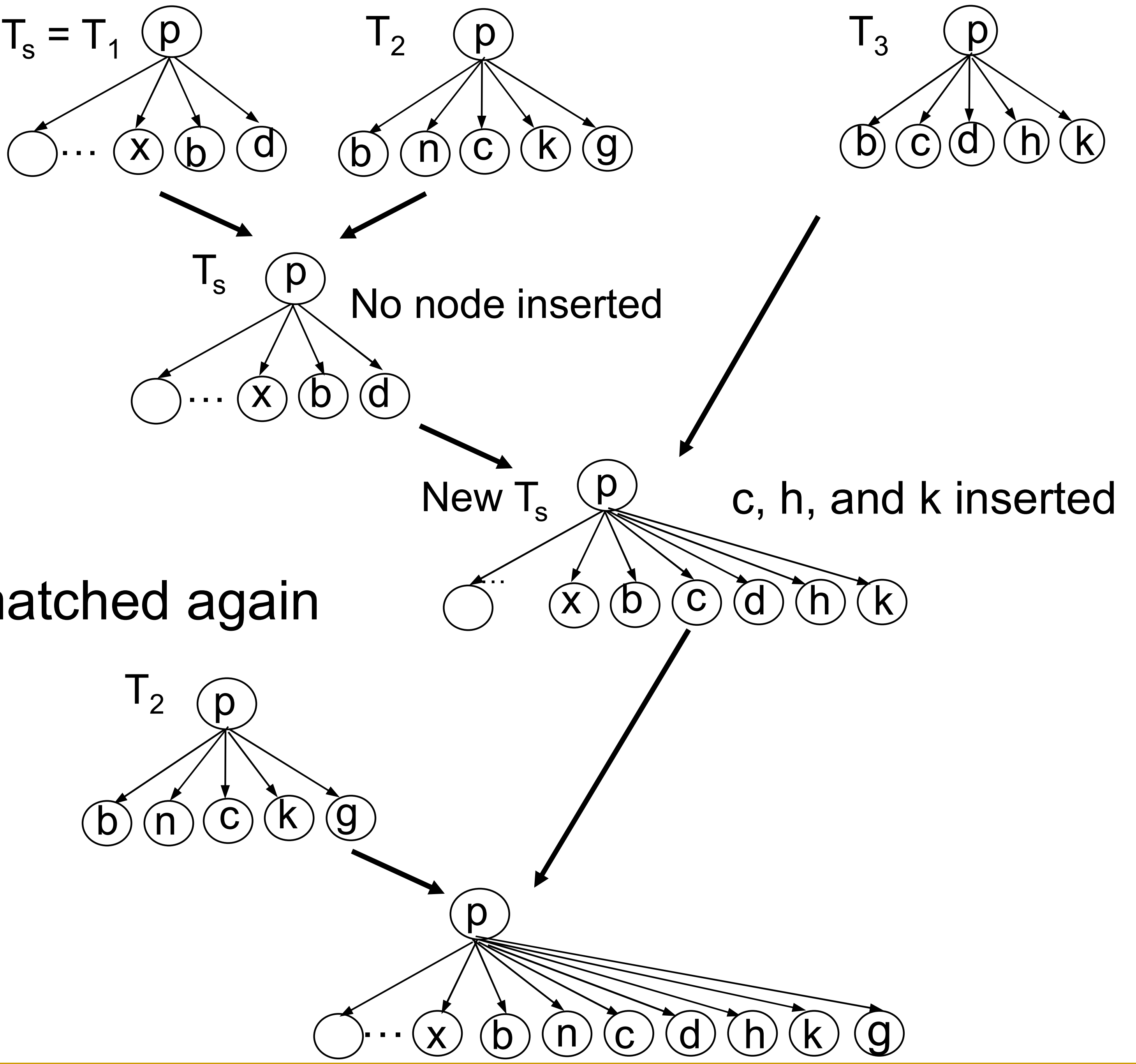
Partial alignment of two trees



```
Algorithm PartialTreeAlignment($S$)
1.  Sort trees in $S$ in descending order of the number of unaligned data items;
2.  $T_s \leftarrow$ the first tree (which is the largest) and delete it from $S$;
3.  $R \leftarrow \emptyset$;
4.  while ($S \neq \emptyset$) do
5.      $T_i \leftarrow$ select and delete next tree from $S$;   // follow the sorted order
6.      STM($T_s$, $T_i$);                                 // tree matching
7.      AlignTrees($T_s$, $T_i$);                          // based on the result from line 6
8.      if $T_i$ is not completely aligned with $T_s$ then
9.          if InsertIntoSeed($T_s$, $T_i$) then          // True: some insertions are done
10.             $S = S \cup R$;
11.             $R \leftarrow \emptyset$
12          endif;
13.         if there are still unaligned items in $T_i$ that are not inserted into $T_s$ then
14.             $R \leftarrow R \cup \{T_i\}$
15.         endif;
16.     endif;
17. endwhile;
18. Output data fields from each $T_i$ to a data table based on the alignment results.
```

Fig. 21. The partial tree alignment algorithm

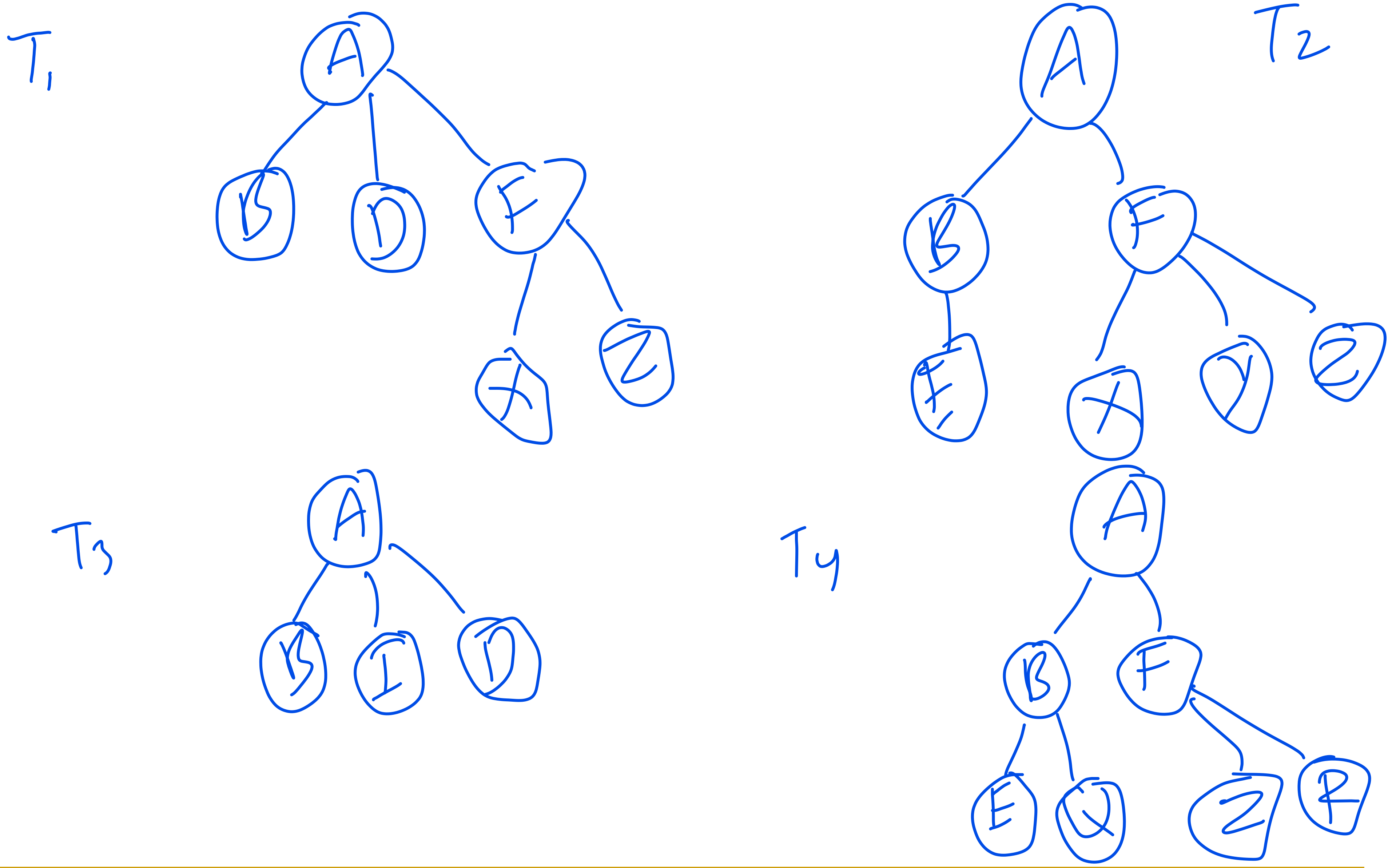
A complete example



Output Data Table

	…	x	b	n	c	d	h	k	g
T_1	…	1	1			1			
T_2			1	1	1			1	1
T_3			1		1	1	1	1	

Example: Align the following trees



Road map

- Introduction
- Data Model and HTML encoding
- Wrapper induction
- Automatic Wrapper Generation: Two Problems
- String Matching and Tree Matching
- Multiple Alignments
- **Building DOM Trees**
- Extraction Given a List Page: Flat Data Records
- Extraction Given a List Page: Nested Data Records
- Extraction Given Multiple Pages
- Summary

Building DOM trees

- We now start to talk about actual data extraction.
- The usual first step is to build a DOM tree (tag tree) of a HTML page.
  - Most HTML tags work in pairs. Within each corresponding tag-pair, there can be other pairs of tags, resulting in a nested structure.
  - Building a DOM tree from a page using its HTML code is thus natural.
- In the tree, each pair of tags is a **node**, and the nested tags within it are the **children** of the node.

Two steps to build a tree

- **HTML code cleaning**:
  - Some tags do not require closing tags (e.g., <li>, <hr> and <p>) although they have closing tags.
  - Additional closing tags need to be inserted to ensure all tags are balanced.
  - Ill-formatted tags need to be fixed. One popular program is called **Tidy**, which can be downloaded from http://tidy.sourceforge.net/.
- **Tree building**: simply follow the nested blocks of the HTML tags in the page to build the DOM tree. It is straightforward.

Building tree using tags & visual cues

- Correcting errors in HTML can be hard.
- There are also dynamically generated pages with scripts.
- Visual information comes to the rescue.
- As long as a browser can render a page correct, a tree can be built correctly.
    - Each HTML element is rendered as a rectangle.
    - Containments of rectangles representing nesting.

An example

		left	right	top	bottom
1	`<table>`	100	300	200	400
2	`  <tr>`	100	300	200	300
3	`    <td>…</td>`	100	200	200	300
4	`    <td>…</td>`	200	300	200	300
5	`  </tr>`				
6	`  <tr>`	100	300	300	400
7	`    <td>…</td>`	100	200	300	400
8	`    <td>…</td>`	200	300	300	400
9	`  </tr>`				
10	`</table>`				

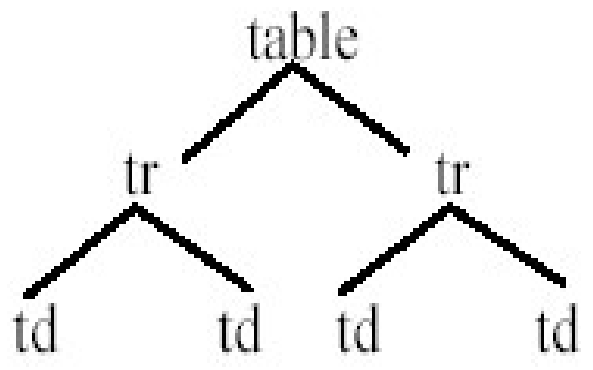


Fig. 23. A HTML code segment, boundary coordinates and the resulting tree