Similarity Search

Trees

Nikolaus Augsten
nikolaus.augsten@sbg.ac.at
Department of Computer Sciences
University of Salzburg

database research group
http://dbresearch.uni-salzburg.at

WS 2019/20

Version November 13, 2019

Outline

Outline

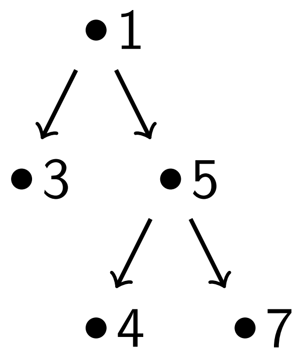
1. What is a Tree?
2. Encoding XML as Trees

What is a Tree?

- Graph: a pair (N, E) of nodes N and edges E between nodes of N
- Tree: a directed, acyclic graph T
  - that is connected and
  - no node has more than one incoming edge
- Edges: $E(\mathrm{T})$ are the edges of T
  - an edge $(\mathrm{p}, \mathrm{c}) \in E(\mathrm{T})$ is an ordered pair
  - with $\mathrm{p}, \mathrm{c} \in N(\mathrm{T})$
- "Special" Nodes: $N(\mathrm{T})$ are the nodes of T
  - parent/child: $(\mathrm{p}, \mathrm{c}) \in E(T) \Leftrightarrow$ p is the parent of c, c is the child of p
  - siblings: c_1 and c_2 are siblings if they have the same parent node
  - root node: node without parent (no incoming edge)
  - leaf node: node without children (no outgoing edge)
  - fanout: fanout f_{v} of node v is the number of children of v

Unlabeled Trees

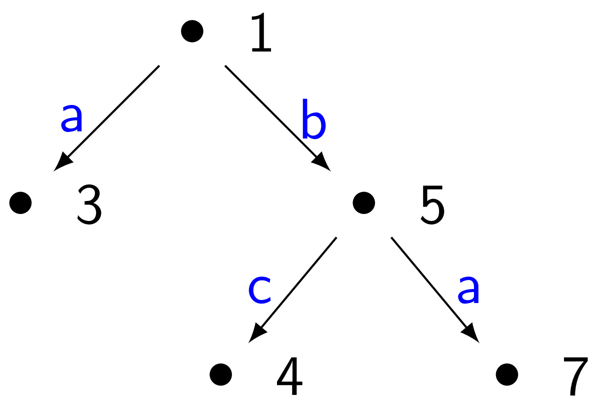
- Unlabeled Tree:
  - the focus is on the structure, not on distinguishing nodes
  - however, we need to distinguish nodes in order to define edges
    $\Rightarrow$ each node v has a unique identifier id(v) within the tree
- Example: $T = (\{1, 3, 5, 4, 7\}, \{(1, 3), (1, 5), (5, 4), (5, 7)\})$



Edge Labeled Trees

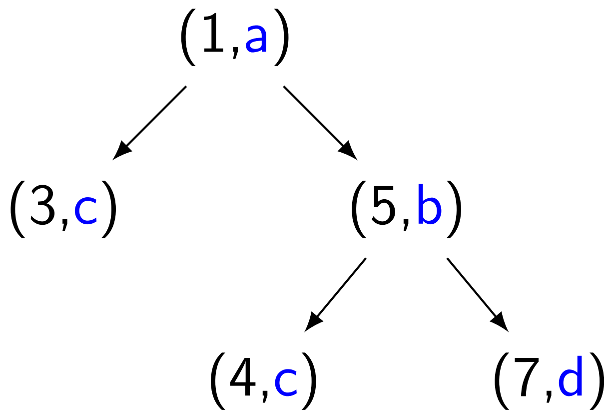
- Edge Labeled Tree:
  - an edge $e \in E(\mathrm{T})$ between nodes a and b is a triple $e = (\mathrm{id(a)}, \mathrm{id(b)}, \lambda(e))$
  - id(a) and id(b) are node IDs
  - $\lambda(e)$ is the edge label (not necessarily unique within the tree)
- Example:

$$\mathrm{T} = (\{1,3,5,4,7\}, \{(1,3,\mathtt{a}), (1,5,\mathtt{b}), (5,4,\mathtt{c}), (5,7,\mathtt{a})\})$$



Node Labeled Trees

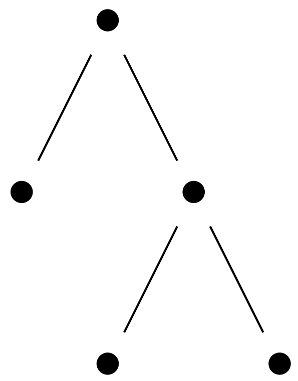
- Node Labeled Tree:
  - a node v $\in N(\mathrm{T})$ is a pair $(\mathrm{id}(v), \lambda(v))$
  - id(v) is unique within the tree
  - label $\lambda(v)$ needs not to be unique
- Intuition:
  - The identifier is the key of the node.
  - The label is the data carried by the node.
- Example: $\mathrm{T} = (\{(1, \mathtt{a}), (3, \mathtt{c}), (5, \mathtt{b}), (4, \mathtt{c}), (7, \mathtt{d})\},$
  $\{(1, 3), (1, 5), (5, 4), (5, 7)\})$



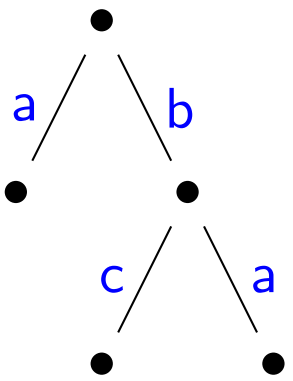
Notation and Graphical Representation

- Notation:
  - node identifiers: $\text{id}(v_i) = i$
  - tree identifiers: $T_1, T_2, \ldots$
- Graphical representation
  - we omit brackets for (identifier,label)-pairs
  - we (sometimes) omit node identifiers at all
  - we do not show the direction of edges
    (edges are always directed from root to leave)

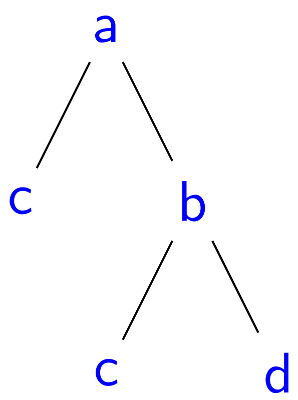
unlabeled tree



edge labeled tree



node labeled tree



Ordered Trees

- Ordered Trees: siblings are ordered
- contiguous siblings $s_1 < s_2$ have no sibling x such that $s_1 < x < s_2$
- c_i is the *i*-th child of p if
  - p is the parent of c_i, and
  - $i = |\{x \in N(T) : (p, x) \in E(T), x \leq c_i\}|$
- Example:

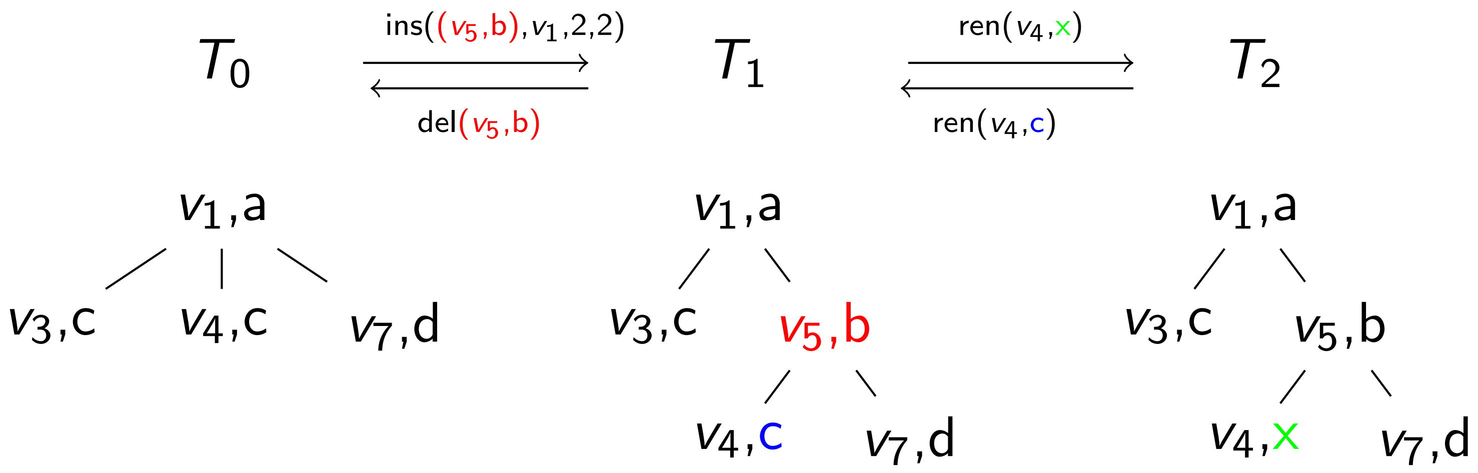
```
 Unordered Trees          |       Ordered Trees
   a            a         |      a            a
  /|\          /|\        |     /|\          /|\
 c b d   =    d b c       |    c b d   ≠    d b c
    / \      / \          |       / \      / \
   e   f    f   e         |      e   f    f   e
```

- Note: “ordered” does not necessarily mean “sorted alphabetically”

Edit Operations

- We assume ordered, labeled trees
- Rename node: $ren(\mathrm{v}, l')$
  - change label l of v to $l' \neq l$
- Delete node: $del(\mathrm{v})$ (v is not the root node)
  - remove v
  - connect v's children directly to v's parent node (preserving order)
- Insert node: $ins(\mathrm{v}, \mathrm{p}, k, m)$
  - remove m consecutive children of p, starting with the child at position k, i.e., the children $\mathrm{c}_k, \mathrm{c}_{k+1}, \ldots, \mathrm{c}_{k+m-1}$
  - insert $\mathrm{c}_k, \mathrm{c}_{k+1}, \ldots, \mathrm{c}_{k+m-1}$ as children of the new node v (preserving order)
  - insert new node v as k-th child of p
- Insert and delete are inverse edit operations (i.e., insert undoes delete and vice versa)

Example: Edit Operations



Outline

Representing XML as a Tree

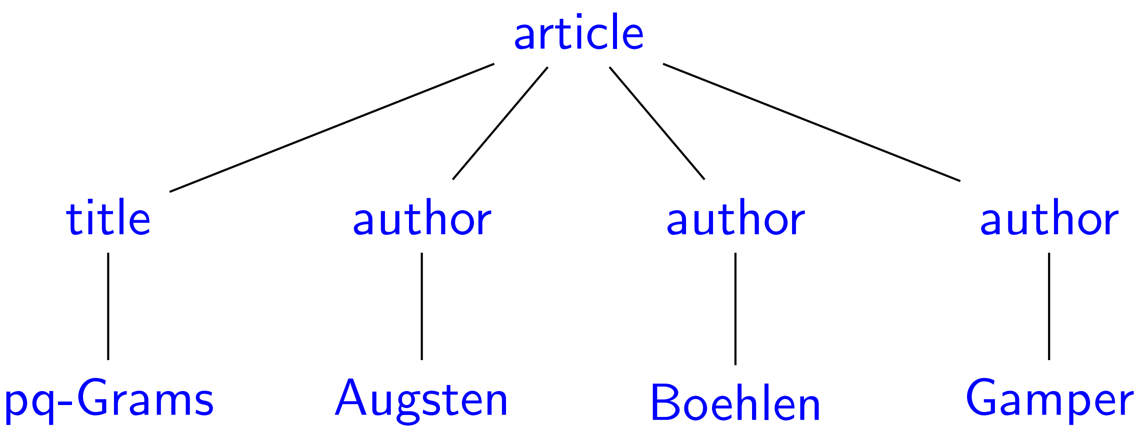
- Many possibilities – we will consider
  - single-label tree
  - double-label tree
- Pros/cons depend on application!

XML as a Single-Label Tree

- The XML document is stored as a tree with:
  - XML element: node labeled with element tag name
  - XML attribute: node labeled with attribute name
  - Text contained in elements/attributes: node labeled with the text-value
- Element nodes contain:
  - nodes of their sub-elements
  - nodes of their attributes
  - nodes with their text values
- Attribute nodes contain:
  - single node with their text value
- Text nodes are always leaves
- Order:
  - sub-element and text nodes are ordered
  - attributes are not ordered (approach: store them before all sub-elements, sort according to attribute name)

Example: XML as a Single-Label Tree

```
<article title='pq-Grams'>
  <author>Augsten</author>
  <author>Boehlen</author>
  <author>Gamper</author>
</article>
```

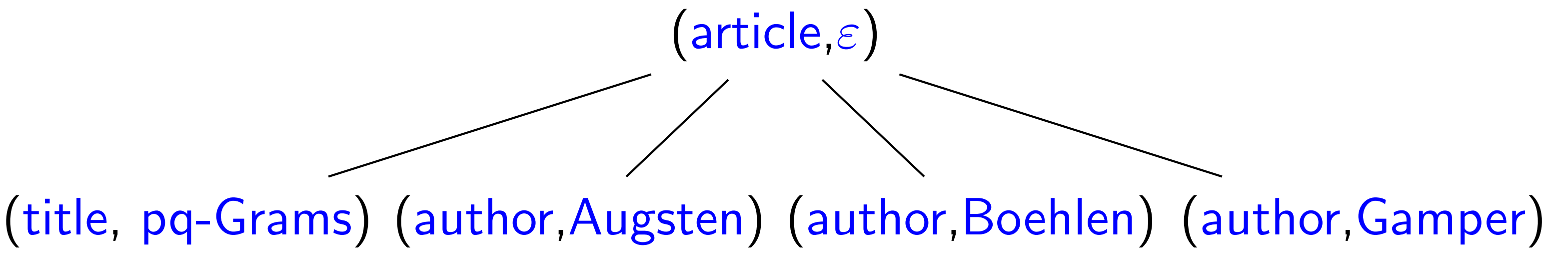


XML as a Double-Label Tree

- Node labels are pairs
- The XML document is stored as a tree with:
  - XML element: node labeled with (tag-name,text-value)
  - XML attribute: node labeled with (attribute-name,text-value)
- Element nodes contain:
  - nodes of their sub-elements and attributes
- Attribute nodes are always leaves
- Element nodes without attributes or sub-elements are leaves
- Order:
  - sub-element nodes are ordered
  - attributes are not ordered (approach: see previous slide)
- Limitation: Can represent
  - *either* elements with sub-elements and/or attributes
  - *or* elements with a text value

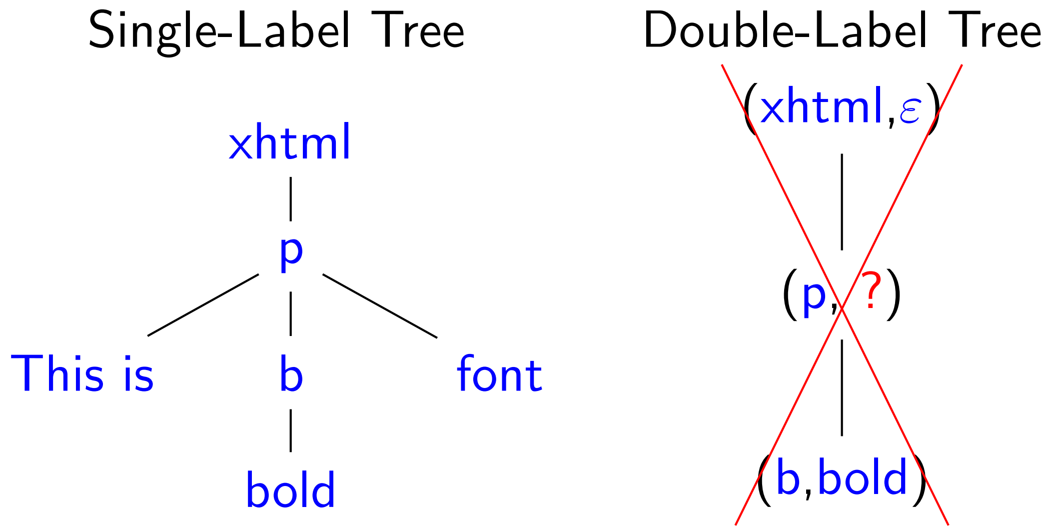
Example: XML as a Double-Label Tree

```
<article title='pq-Grams'>
  <author>Augsten</author>
  <author>Boehlen</author>
  <author>Gamper</author>
</article>
```



Example: Single- vs. Double-Label Tree

```
<xhtml>
  <p>This is <b>bold</b> font.</p>
<xhtml>
```



Parsing XML

We discuss two popular parsers for XML:

- DOM – Document Object Model
- SAX – Simple API for XML

DOM – Document Object Model

- W3C[1] standard for accessing and manipulating XML documents
- Tree-based: represents an XML document as a tree (single-label tree with additional node info, e.g. node type)
- Elements, attributes, and text values are nodes
- DOM parsers load XML into main memory
  - random access by traversing tree :-)
  - large XML documents do not fit into main memory :-(

[1] http://www.w3schools.com/dom

SAX – Simple API for XML

- “de facto” standard for parsing XML[2]
- Event-based: reports parsing events (e.g., start and end of elements)
  - no random access :-(
  - you see only one element/attribute at a time
  - you can parse (arbitrarily) large XML documents :-)
- Java API available for both, DOM and SAX
- For importing XML into a database: use SAX!

[2] `http://www.saxproject.org`