# Similarity Search

## Trees

Nikolaus Augsten

nikolaus.augsten@plus.ac.at
Department of Computer Science
University of Salzburg

https://dbresearch.uni-salzburg.at

WS 2026/27

Version September 14, 2026

# Outline

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

# Outline

1 What is a Tree?

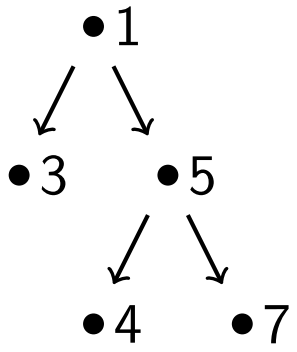
2 Encoding XML as Trees

# What is a Tree?

- Graph: a pair $(N, E)$ of nodes $N$ and edges $E$ between nodes of $N$
- Tree: a finite, 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: $\mathrm{c}_1$ and $\mathrm{c}_2$ 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 $f_{\mathrm{v}}$ of node v is the number of children of v
  - descendants: node x is a descendant of v if x is reachable from v

# Unlabeled Trees

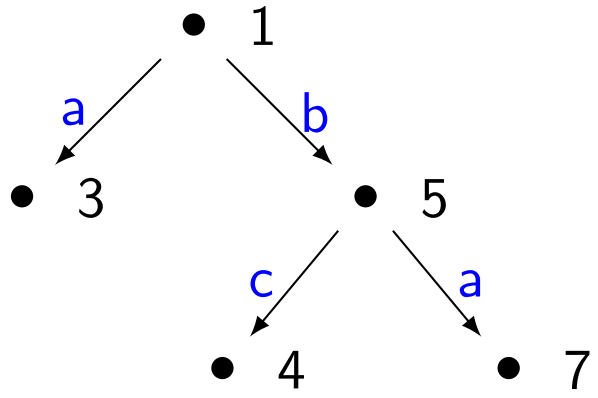
- Unlabeled Tree:
  - the focus is on the structure, not on distinguishing nodes
  - however, we need to distinguish nodes in order to define edges
    $\Rightarrow$ each node v has a unique identifier id(v) within the tree
- Example: $T = (\{1, 3, 5, 4, 7\}, \{(1, 3), (1, 5), (5, 4), (5, 7)\})$



# Edge Labeled Trees

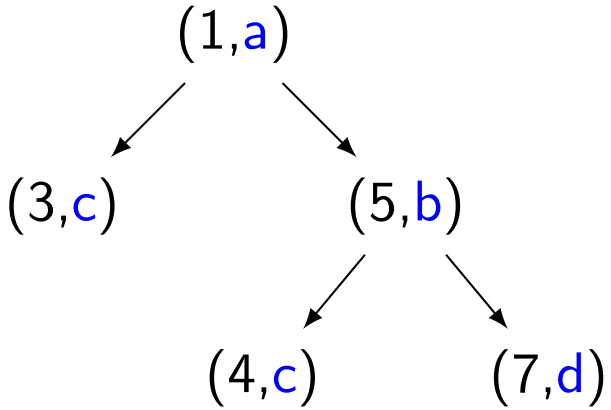
- Edge Labeled Tree:
  - an edge $e \in E(\mathsf{T})$ between nodes a and b is a triple $e = (\mathrm{id}(\mathsf{a}), \mathrm{id}(\mathsf{b}), \lambda(e))$
  - id(a) and id(b) are node IDs
  - $\lambda(e)$ is the edge label (not necessarily unique within the tree)
- Example:

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



# Node Labeled Trees

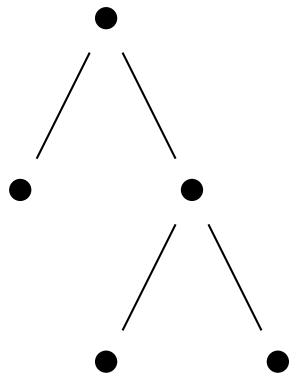
- Node Labeled Tree:
  - a node v $\in N(T)$ is a pair $(id(v), \lambda(v))$
  - id(v) is unique within the tree
  - label $\lambda(v)$ needs not to be unique
- Intuition:
  - The identifier is the key of the node.
  - The label is the data carried by the node.
- Example: $T = (\{(1, a), (3, c), (5,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



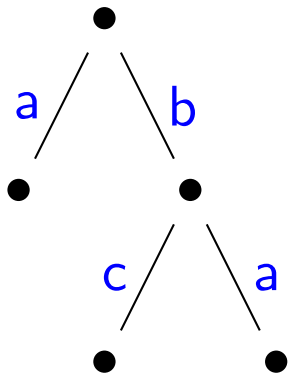
# Notation and Graphical Representation

- Notation:
  - node identifiers: $\mathrm{id}(v_i) = i$
  - 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)

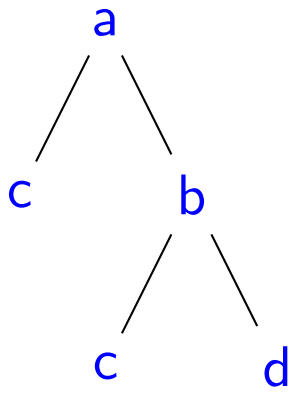
**unlabeled tree**



**edge labeled tree**



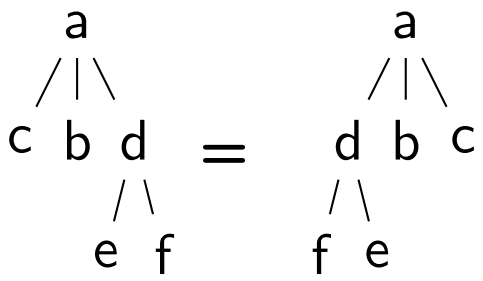

**node labeled tree**



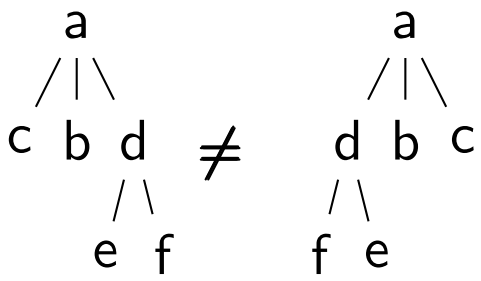
# Ordered Trees

- 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
  - $i = |\{x \in N(T) : (p, x) \in E(T), x \leq c_i\}|$
- Node order $<$ is a strict total order that respects the tree structure:
  (a) If $c$ is a child of $v$, then $c < v$
  (b) If $f < g$ and $f$ is not a descendant of $g$, then $f < g'$ for all descendants $g'$ of $g$
- Example:

**Unordered Trees**




**Ordered Trees**

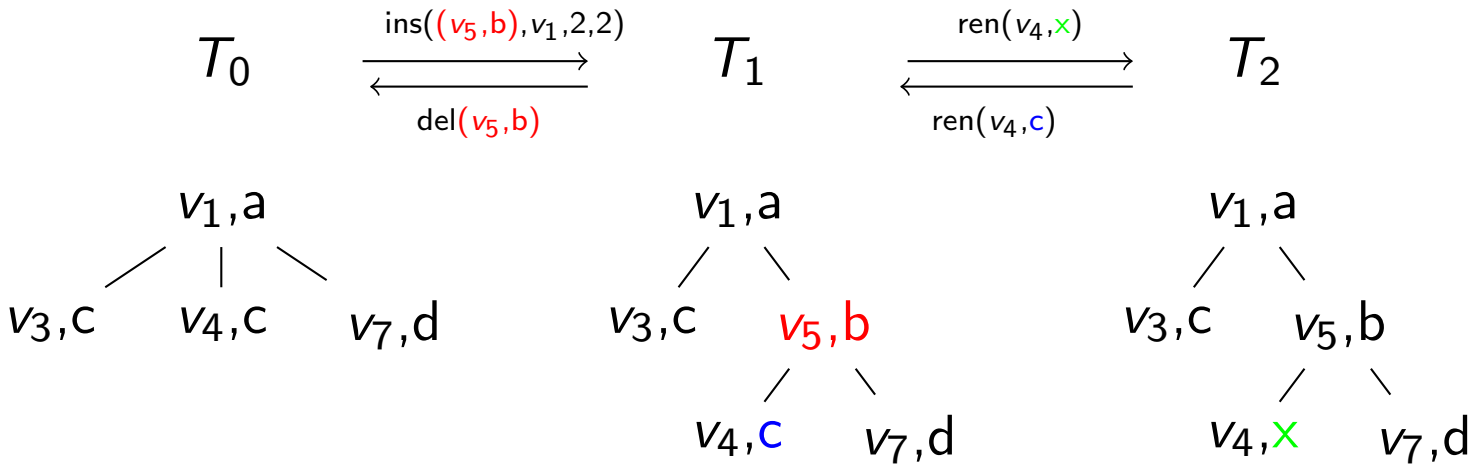



- Note: "ordered" does not necessarily mean "sorted alphabetically"

# Edit Operations

- We assume ordered, labeled trees
- Rename node: $ren(v, l')$
  - change label $l$ of v to $l' \neq l$
- Delete node: $del(v)$ (v is not the root node)
  - remove v
  - connect v's children directly to v's parent node (preserving order)
- Insert node: $ins(v, p, k, m)$
  - detach $m$ consecutive children of p, starting with the child at position $k$, i.e., the children $c_k, c_{k+1}, \ldots, c_{k+m-1}$
  - attach $c_k, c_{k+1}, \ldots, c_{k+m-1}$ 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

1 What is a Tree?

2 Encoding XML as Trees

# Representing XML as a Tree

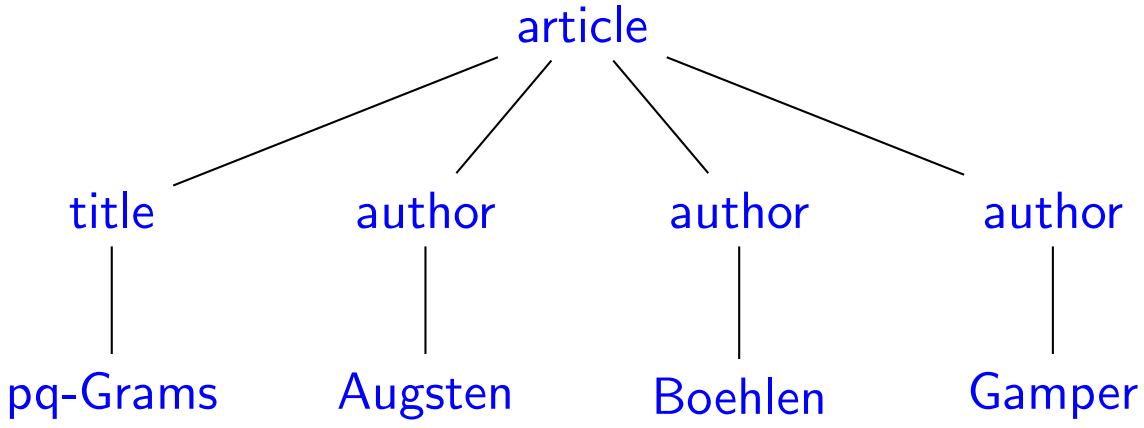
- Many possibilities – we will consider
  - single-label tree
  - double-label tree
- Pros/cons depend on application!

# XML as a Single-Label Tree

- The XML document is encoded 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>
```

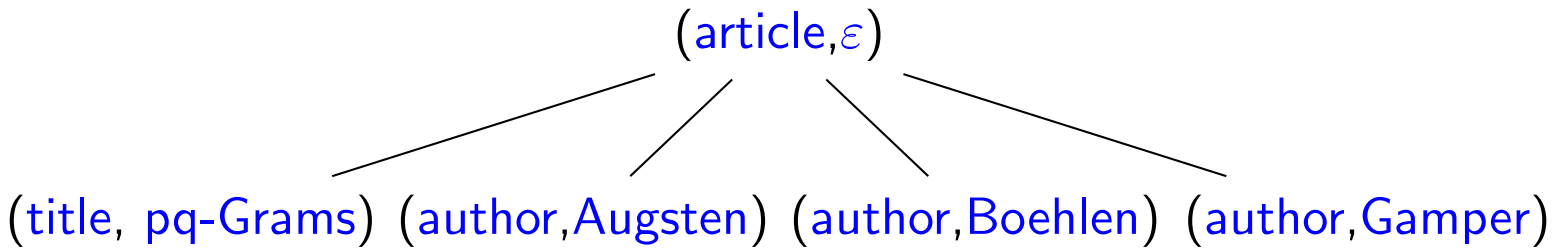


# XML as a Double-Label Tree

- Node labels are pairs
- The XML document is encoded 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

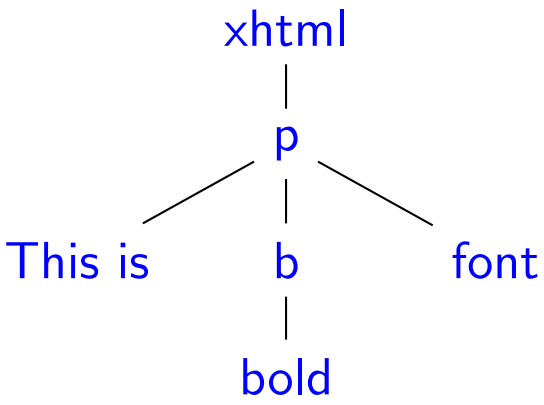
```
<article title='pq-Grams'>
  <author>Augsten</author>
  <author>Boehlen</author>
  <author>Gamper</author>
</article>
```



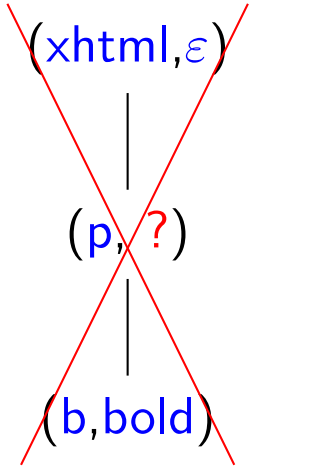
# Example: Single- vs. Double-Label Tree

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

Single-Label Tree




Double-Label Tree



# Parsing XML

We discuss two popular parsers for XML:

- DOM – Document Object Model
- SAX – Simple API for XML
- StAX – Streaming 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 / StAX

- SAX – Simple API for XML[2]
  - 'de facto" standard for parsing XML
- 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 :-)
- StAX - Streaming API for XML[3]
  - similar to SAX, but pull-based (vs. SAX: push)
  - pull: the client receives the next event on request
- Java API available for DOM, SAX, and StAX.
- For importing XML into a database: use SAX or StAX!

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

[3] `https://en.wikipedia.org/wiki/StAX`