

Similarity Search

Trees

Nikolaus Augsten

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



<http://dbresearch.uni-salzburg.at>

WS 2017/2018

Version March 12, 2018

Outline

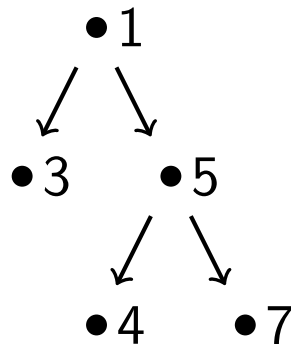
- 1 What is a Tree?

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:** $E(T)$ are the edges of T
 - an edge $(p, c) \in E(T)$ is an ordered pair
 - with $p, c \in N(T)$
- **“Special” Nodes:** $N(T)$ are the nodes of T
 - **parent/child:** $(p, c) \in E(T) \Leftrightarrow p$ 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
 - ⇒ each node v has a unique identifier $\text{id}(v)$ within the tree
- Example: $T = (\{1, 3, 5, 4, 7\}, \{(1, 3), (1, 5), (5, 4), (5, 7)\})$



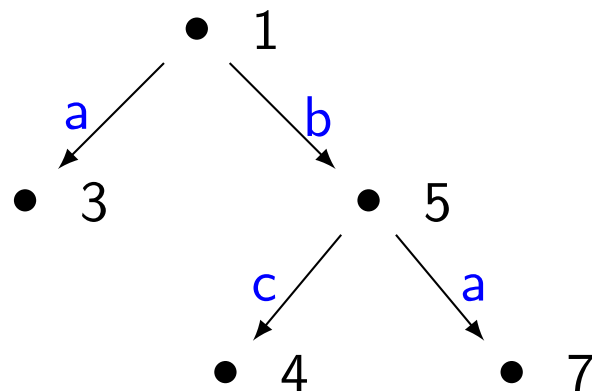
Edge Labeled Trees

- **Edge Labeled Tree:**

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

- **Example:**

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



Node Labeled Trees

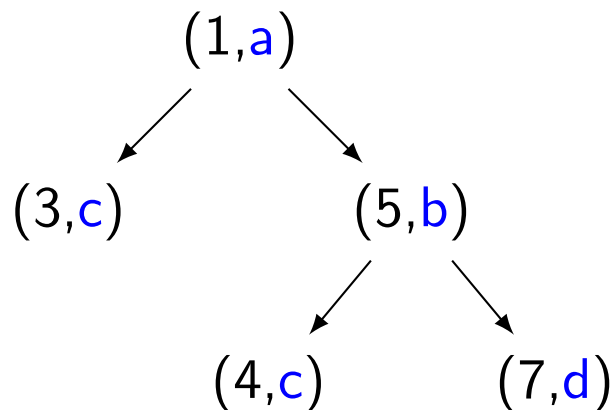
- **Node Labeled Tree:**

- 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.

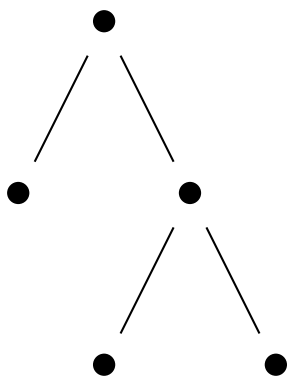
- **Example:** $T = (\{(1, a), (3, c), (5, b), (4, c), (7, d)\}, \{(1, 3), (1, 5), (5, 4), (5, 7)\})$



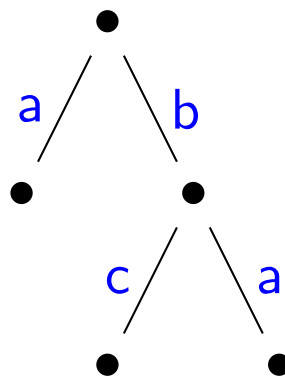
Notation and Graphical Representation

- **Notation:**
 - node identifiers: $\text{id}(v_i) = i$
 - tree identifiers: T_1, T_2, \dots
- **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 leaf)

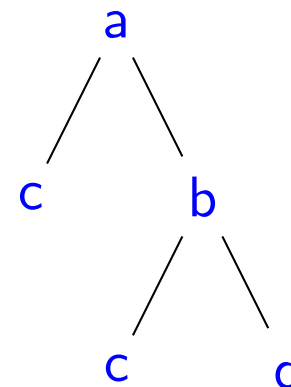
unlabeled tree



edge labeled tree



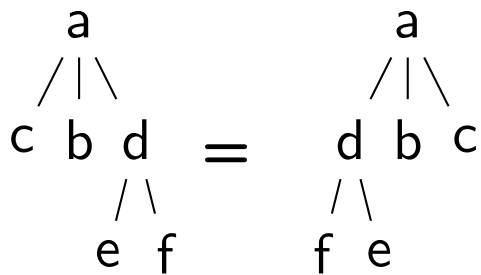
node labeled tree



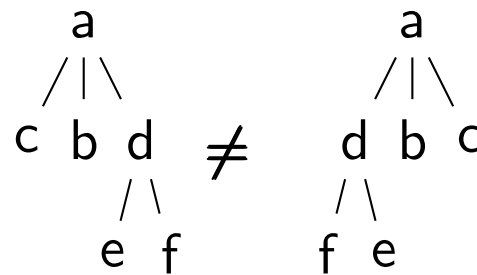
Ordered Trees

- **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



- **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:** $ins(v, p, k, m)$
 - remove m consecutive children of p , starting with the child at position k , i.e., the children $c_k, c_{k+1}, \dots, c_{k+m-1}$
 - insert $c_k, c_{k+1}, \dots, c_{k+m-1}$ 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

