Chapter 8

Tree Algorithms - Keyword Query

Acknowledgements

- Tutorialspoint. Data Structure and Algorithms – Tree. https://www.tutorialspoint.com/data_structures_algorithms/tree_data_structure.htm
- Tree (data structure). Wikipedia. https://en.wikipedia.org/wiki/Tree_(data_structure)
- Node Indexes. Konsolaki Konstantina and Fafalios Pavlos. University of Crete, Department of Computer Science.
- Lin Guo, Feng Shao, Chavdar Botev, Jayavel Shanmugasundaram. XRANK: Ranked Keyword Search over XML Documents. SIGMOD'03.
- Yu Xu, Yannis Papakonstantinou. Efficient keyword search for smallest LCAs in XML databases. SIGMOD'05.
- Evandrino G. Barros, Alberto H.F. Laender, Mirella M. Moro, Altigran S. da Silva. LCA-based algorithms for efficiently processing multiple keyword queries over XML streams. DKE'16.
- Junfeng Zhou, Zhifeng Bao, Wei Wang, Tok Wang Ling, Ziyang Chen, Xudong Lin, Jingfeng Guo. Fast SLCA and ELCA Computation for XML Keyword Queries based on Set Intersection. ICDE'12.
- Zhifeng Bao, Tok Wang Ling, Bo Chen, Jiaheng Lu. Effective XML Keyword Search with Relevance Oriented Ranking. ICDE'09.
- Ziyang Liu, Yi Chen. Identifying meaningful return information for XML keyword search. SIGMOD'07.
- Thuy Ngoc Le, Tok Wang Ling. Survey on Keyword Search over XML Documents. SIGMOD'16.
- Yue Zhao. Shenyang University.

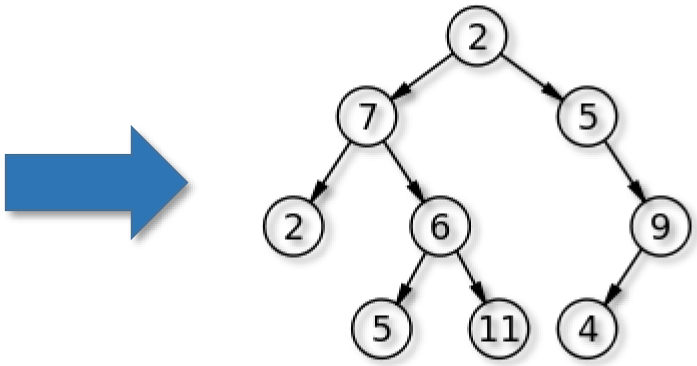
Chapter Outline

- Tree data
- Tree encoding
- Keyword query semantics
- Keyword query processing
- Distributed algorithms

Tree Data

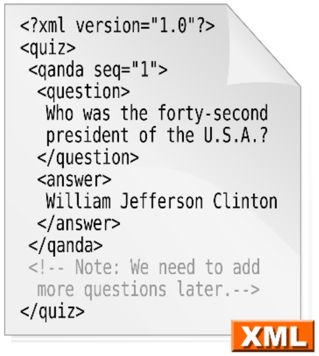
Tree (Data Structure)

- In computer science, a **tree** is a widely used *abstract data type* (ADT)—or *data structure* implementing this ADT
- which simulates a hierarchical tree structure, with a root value and subtrees of children with a parent node, represented as a set of linked nodes



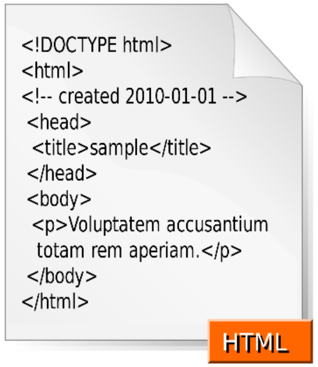
Tree Applications

- Data types
  - XML
  - HTML
  - JSON
- Applications
  - Catalogue
  - Operating System paths
  - Organizations
  - Playoffs (Reverse)



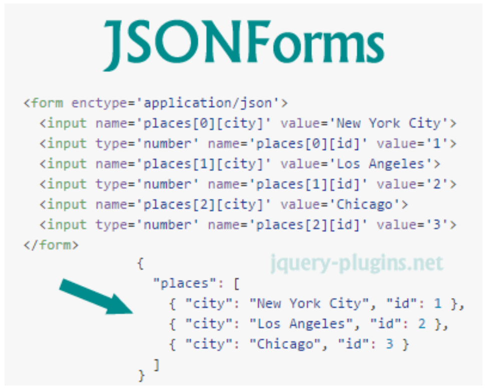
```
<?xml version="1.0"?>
<quiz>
 <qanda seq="1">
  <question>
   Who was the forty-second
   president of the U.S.A.?
  </question>
  <answer>
   William Jefferson Clinton
  </answer>
 </qanda>
 <!-- Note: We need to add
  more questions later.-->
</quiz>
```

XML



```
<!DOCTYPE html>
<html>
<!-- created 2010-01-01 -->
 <head>
  <title>sample</title>
 </head>
 <body>
  <p>Voluptatem accusantium
  totam rem aperiam.</p>
 </body>
</html>
```

HTML



JSONForms

```
<form enctype='application/json'>
  <input name='places[0][city]' value='New York City'>
  <input type='number' name='places[0][id]' value='1'>
  <input name='places[1][city]' value='Los Angeles'>
  <input type='number' name='places[1][id]' value='2'>
  <input name='places[2][city]' value='Chicago'>
  <input type='number' name='places[2][id]' value='3'>
</form>
```

```
{
  "places": [
    { "city": "New York City", "id": 1 },
    { "city": "Los Angeles", "id": 2 },
    { "city": "Chicago", "id": 3 }
  ]
}
```

jquery-plugins.net

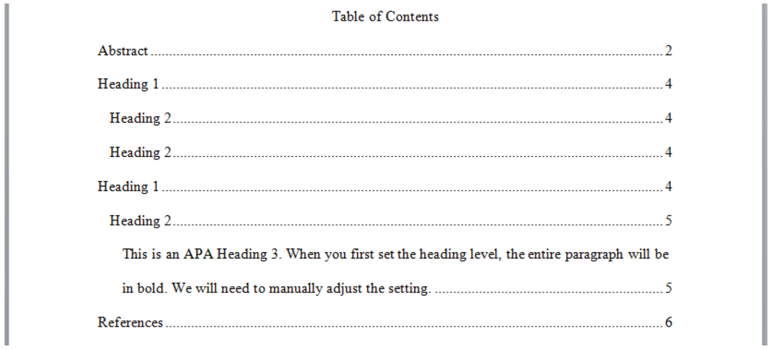
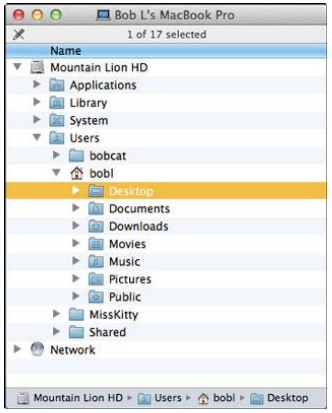
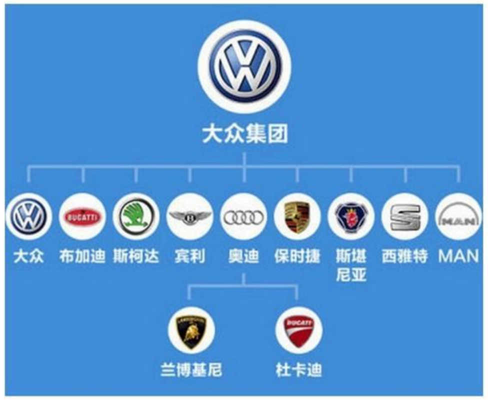


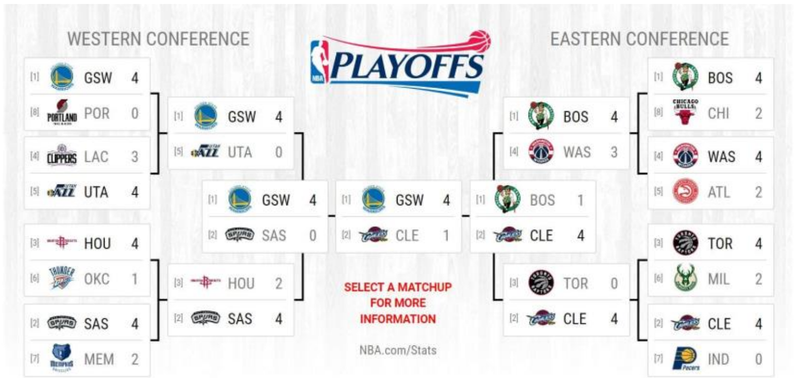
Table of Contents



```
Bob L's MacBook Pro
1 of 17 selected
Name
▼ Mountain Lion HD
  ▶ Applications
  ▶ Library
  ▶ System
  ▼ Users
    ▶ bobcat
    ▼ bobl
      ▶ Desktop
      ▶ Documents
      ▶ Downloads
      ▶ Movies
      ▶ Music
      ▶ Pictures
      ▶ Public
    ▶ MissKitty
    ▶ Shared
▶ Network

Mountain Lion HD ▸ Users ▸ bobl ▸ Desktop
```



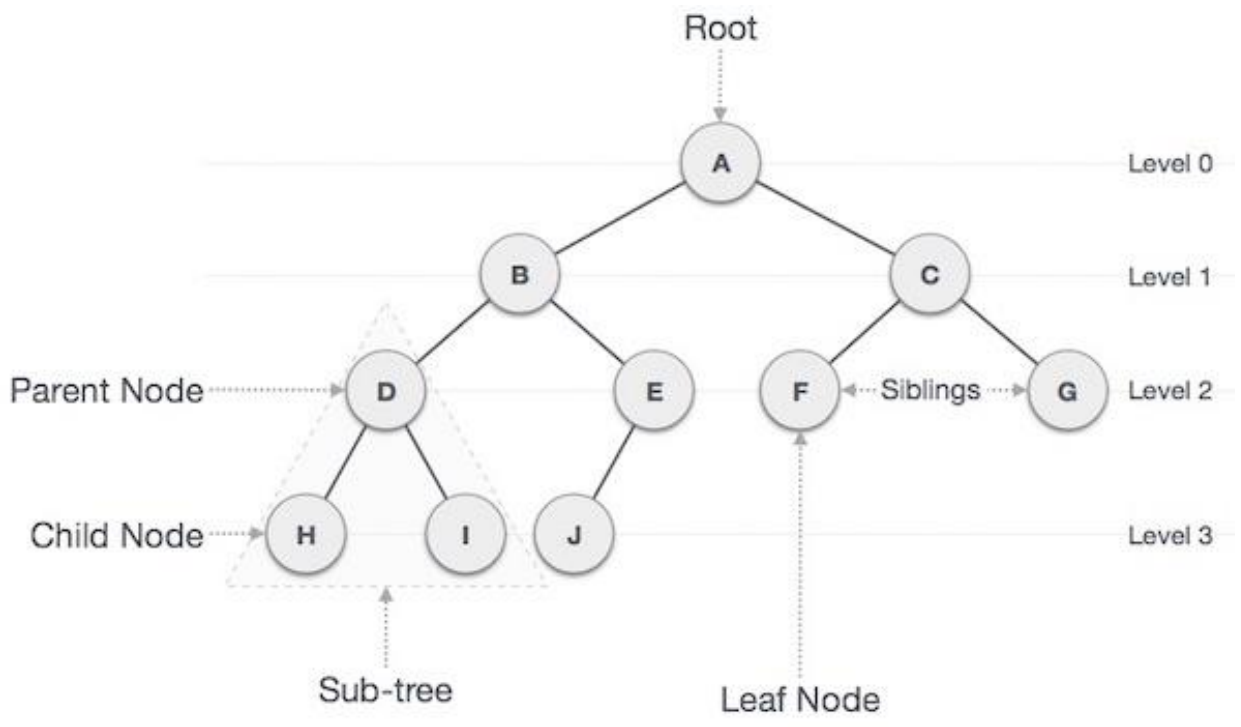


Terminology

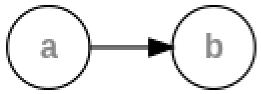
- **Root**: The top node in a tree.
- **Child**: A node directly connected to another node when moving away from the Root.
- **Parent**: The converse notion of a child.
- **Siblings**: A group of nodes with the same parent.
- **Descendant**: A node reachable by repeated proceeding from parent to child.
- **Ancestor**: A node reachable by repeated proceeding from child to parent.
- **Leaf**: (less commonly called External node) A node with no children.
- **Branch** / **Internal node**: A node with at least one child.
- **Degree**: The number of subtrees of a node.
- **Edge**: The connection between one node and another.
- **Path**: A sequence of nodes and edges connecting a node with a descendant.
- **Level**: Defined by 1 + (the number of connections between the node and the root).
- **Height of node**: The number of edges on the longest path between that node and a leaf.
- **Height of tree**: The height of a tree is the height of its root node.
- **Depth**: The depth of a node is the number of edges from the tree's root node to the node.
- **Forest**: A forest is a set of $n \geq 0$ disjoint trees.

Terminology

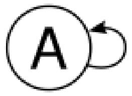
- Root
- Child
- Parent
- Siblings
- Descendant
- Ancestor
- Leaf
- Branch / Internal node
- Degree
- Edge
- Path
- Level
- Height of node
- Height of tree
- Depth
- Forest



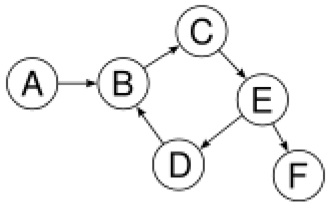
Situations



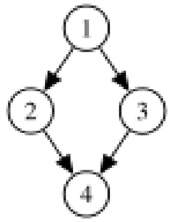
- Each linear list is trivially a tree



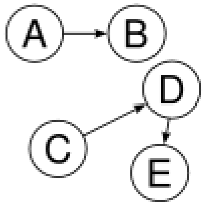
- NOT a tree: cycle A→A. A is the root but it also has a parent



- NOT a tree: cycle B→C→E→D→B. B has more than one parent (inbound edge)



- NOT a tree: undirected cycle 1-2-4-3. 4 has more than one parent (inbound edge).



- NOT a tree: two non-connected parts, A→B and C→D→E. There is more than one root.

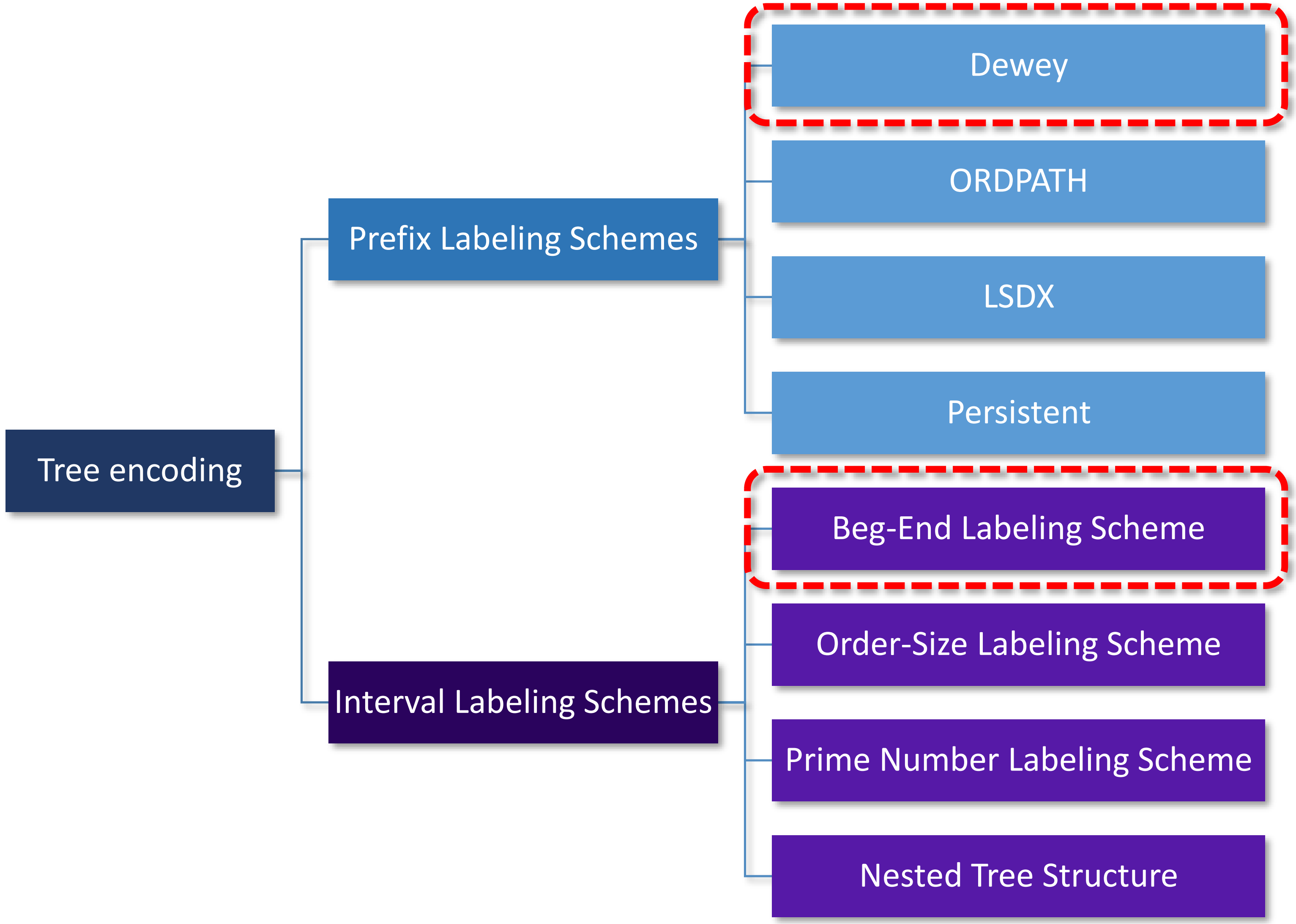
Node Represenation

- E.g., Binary tree

```
struct node {
  int data;
  struct node *leftChild;
  struct node *rightChild;
};
```

Tree Encoding

Variations


Tree encoding
Prefix Labeling Schemes
Interval Labeling Schemes
Dewey
ORDPATH
LSDX
Persistent
Beg-End Labeling Scheme
Order-Size Labeling Scheme
Prime Number Labeling Scheme
Nested Tree Structure

Tree Encoding

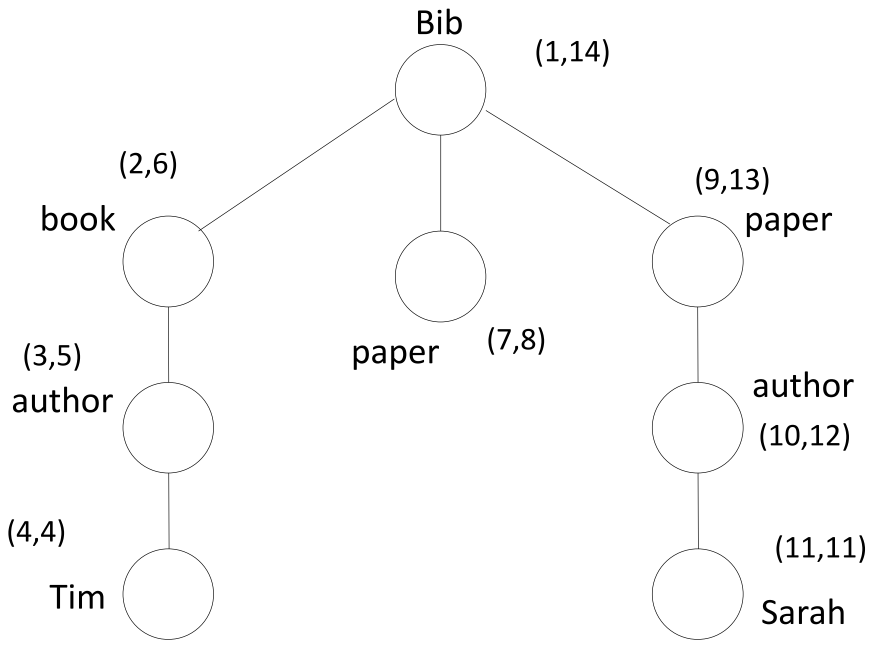
Interval Labeling Schemes

Prefix Labeling Schemes

Beg-End Labeling Scheme

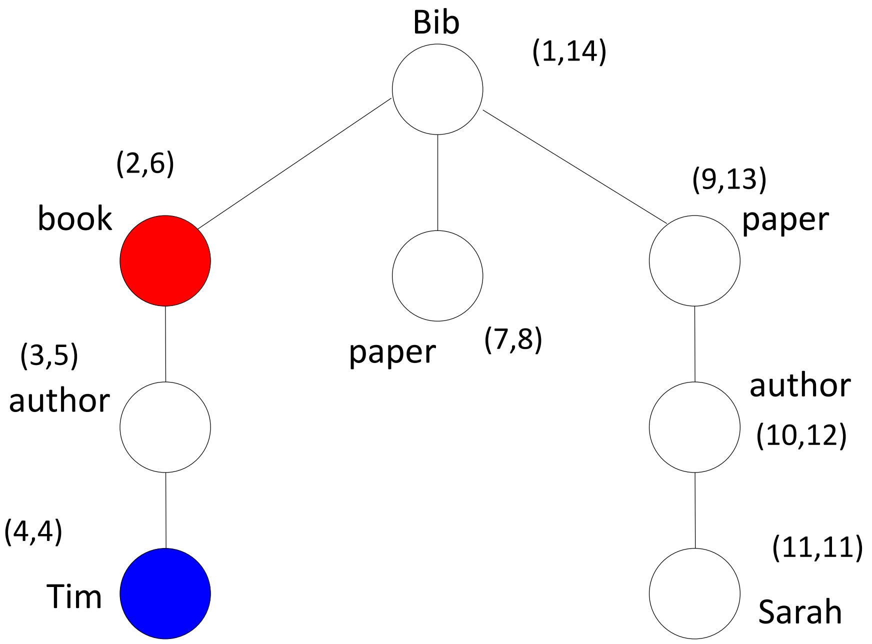
- A pair of numbers is assigned to each node in an XML document according to its sequential traversal order.
  - Starting from the root element, each node is given a “Beg” number.
  - If the end of an attribute, an attribute value, or an ending tag element is reached, the “End” number is assigned. The “End” number is equal to the next sequential number.
    - If the value of the element is a leaf the “Beg” number =“End” number

Example


Bib
(1,14)
(2,6)
book
(9,13)
paper
paper
(7,8)
(3,5)
author
author
(10,12)
(4,4)
Tim
(11,11)
Sarah

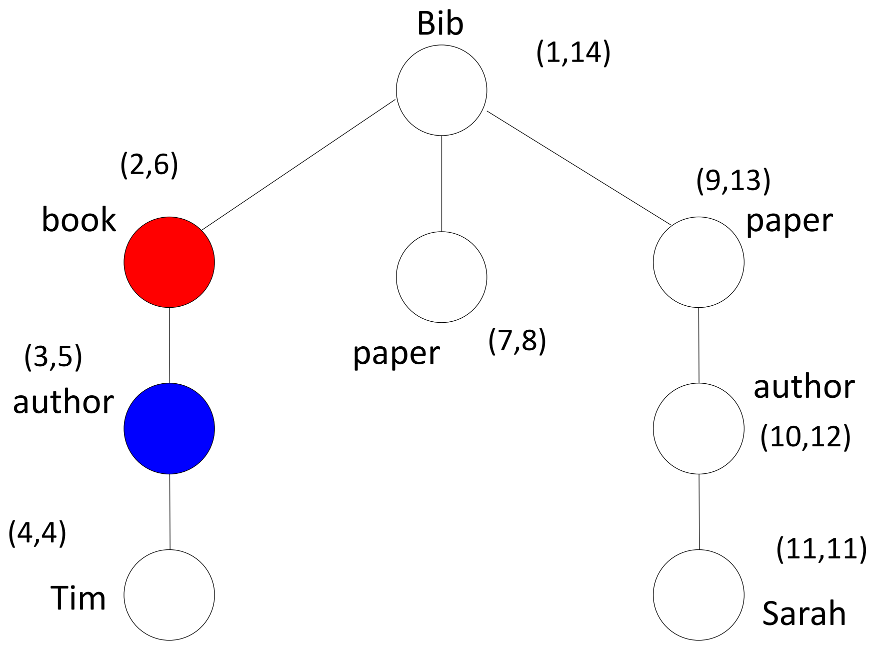
Properties 1

- A “Level” is added to the **(Beg,End)** label to form a node-triplet identification label **(Beg,End,Level)** for each node in the tree.
- **Ancestor-descendant relationship:**
  - In a given data-tree, node “x” is an ancestor of node “y” iff (x.Beg < y.Beg ≤ y.End < x.End) (preorder property).



Properties 2

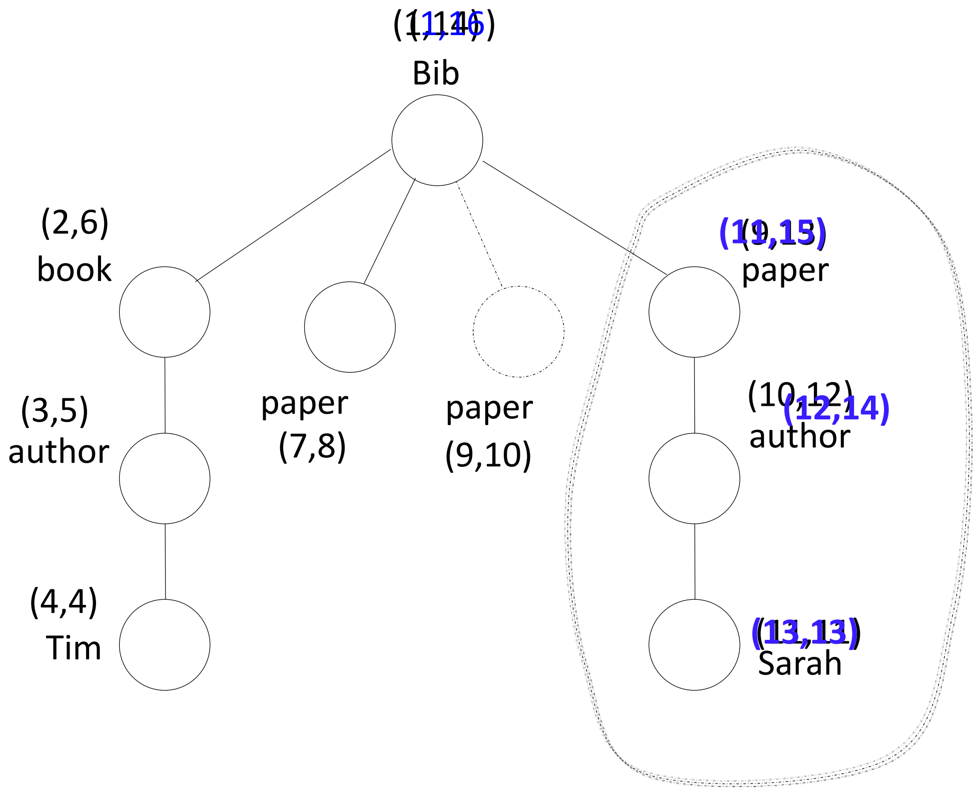
- **Parent-child relationship:**
  - In a given data-tree, node "x" is a parent of node "y" iff (x.Beg < y.Beg ≤ y.End < x.End) and (y.Level = x.Level + 1).
- There is **no way** to locate the siblings of a given node, using only the knowledge of its index numbers.



Are updates possible ?

- Updating the labeling (numbering) scheme of Beg-End is **costly**.
- When a new node is **inserted** into the tree, then all the nodes in the tree, except the left sibling subtrees of the inserted node, have to be updated.
- On the other hand when a node is **deleted** no re-labeling is needed.

Update Example



Tree Encoding

Interval Labeling Schemes

Prefix Labeling Schemes

Dewey Encoding

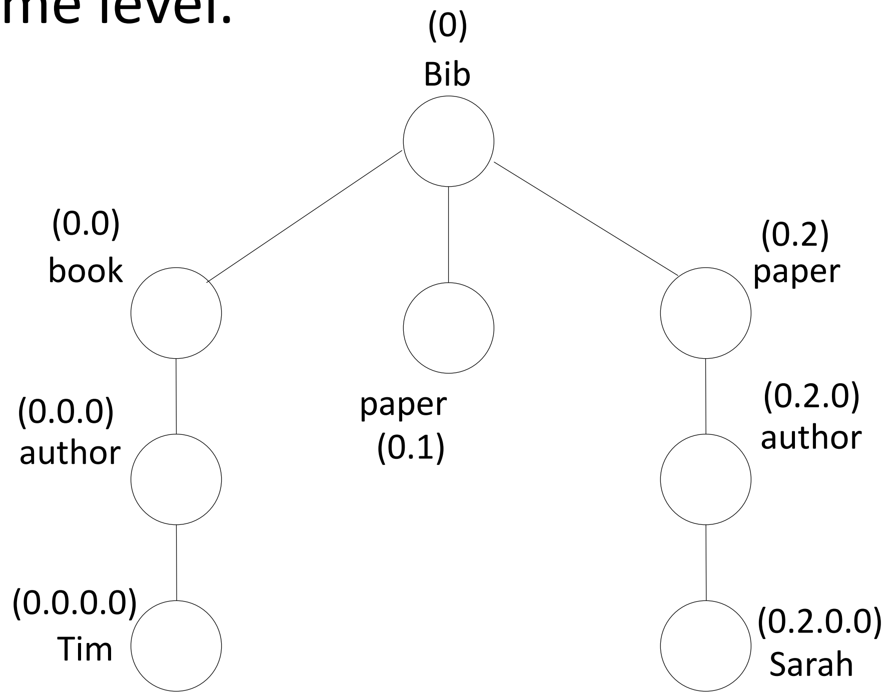
- In a prefix labeling scheme, the label of a node in the XML tree often consists of:
  - A **prefix**, which often represents the label of all the ancestors of the node.
  - A **delimiter**, which in most cases is the fullstop "."
  - A **positional identifier**, which indicates the position of the node relative to its siblings.

Comparison

- Prefix Labeling Schemes:
  - Can **handle updates easier** and more efficient than Interval Labeling Schemes
  - **Support sibling relationship**
- However:
  - Extra space required to store paths
  - Its storage size increases quickly as the depth and the breath of the tree increases
  - Infer a bit more costly ancestor/descendant relationship

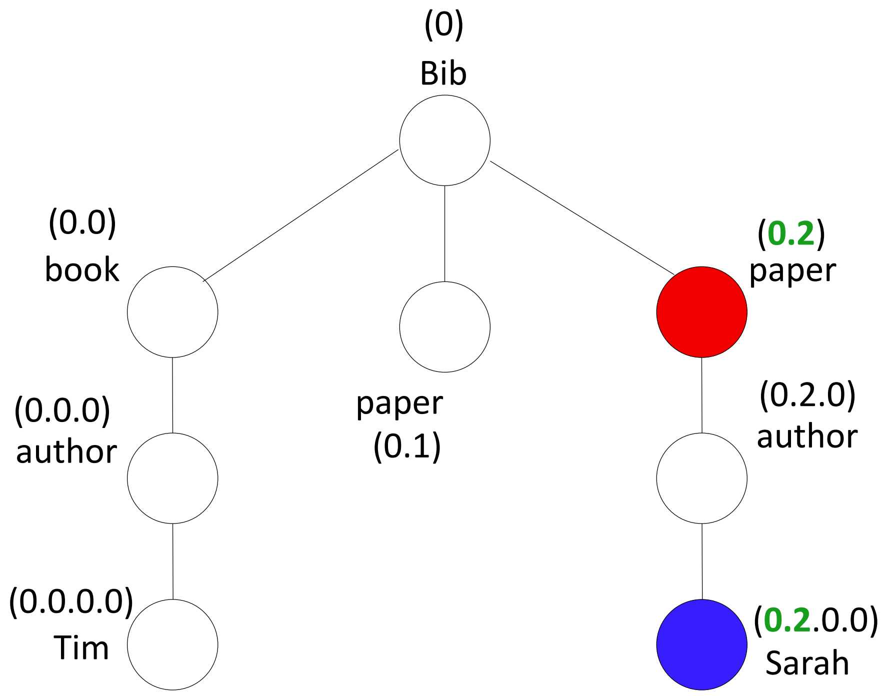
Dewey - Structure

- Each node is assigned a label that represents the path from the document's root to the node.
- Each component of the label represents the local order of an ancestor node.
- Nodes with the same number of delimiters (".") in their label are in the same level.



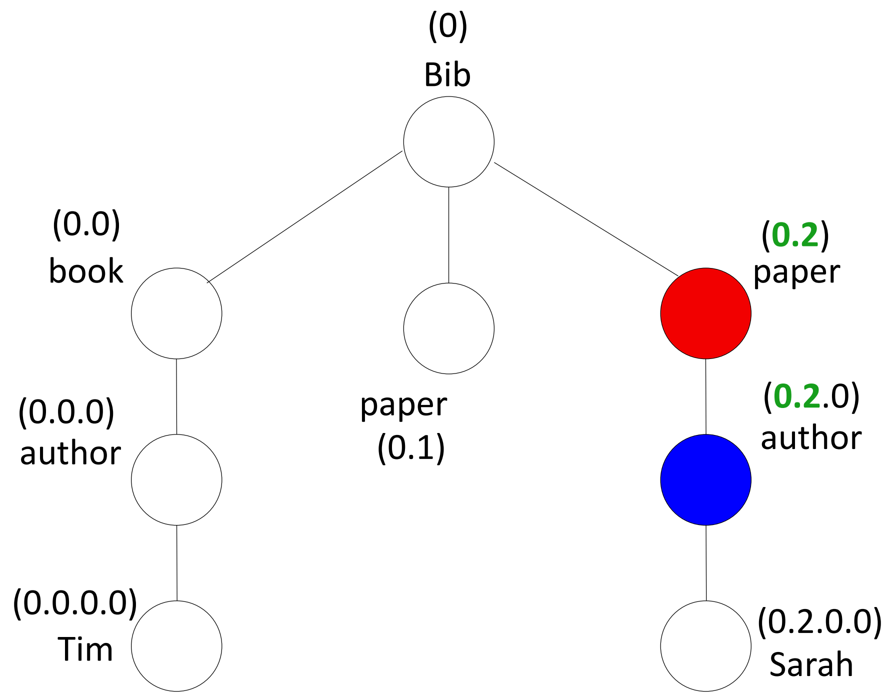
Dewey - Supported Queries

- Ancestors / Descendants
  - Node "X" is an ancestor of node "Y" if the label of node "X" is a **substring** of the label of node "Y".



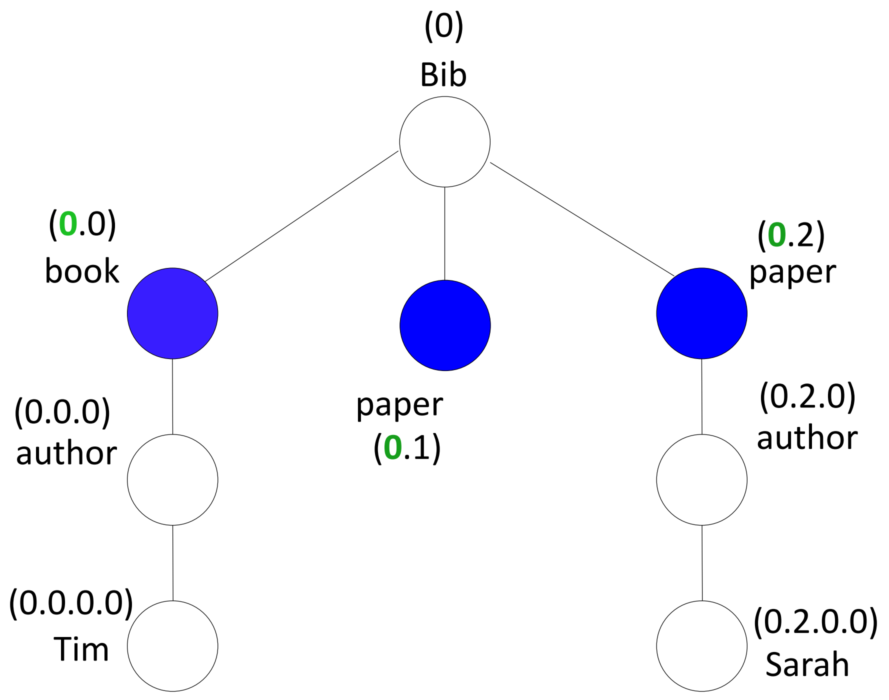
Dewey - Supported Queries

- Parent / Child
  - Node “X” is parent of node “Y” if:
    - The label of node “X” is a substring of the label of node “Y” **and**
    - frags(X) = frags(Y) – 1, where frags(X) is the number of delimiters of the label of node X and frags(Y) is the number of delimiters of label of node Y.



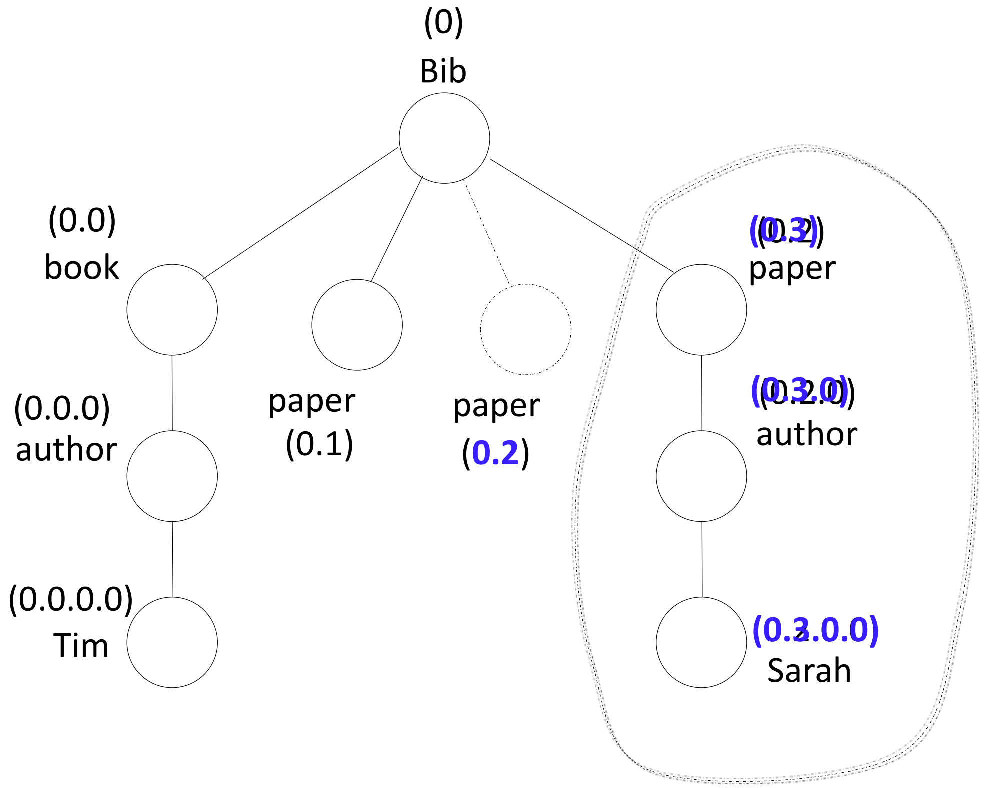
Dewey - Supported Queries

- Siblings
  - Nodes "**X**" and "**Y**" are siblings if:
    - They have the same number of delimiters in their labels **and**
    - X.prefix = Y.prefix, where prefix is the label of the node without its positional identifier



Dewey – Updates

- Insertion of new node
  - The label of the nodes in the subtree rooted at the following sibling need to be updated
  - *O(n)* nodes need relabeling, where *n* is the number of nodes of the XML file



Dewey: Conclusion

- Not efficient for dynamic XML files with many updates
  - Need to re-label many nodes
- As the depth of the tree increases:
  - Label size of a node increases rapidly
    - Storage size increases rapidly
  - It becomes more costly to infer the supported queries between any two nodes (the string prefix matching becomes longer)
- Overflow problem
  - The original fixed length of bits assigned to store the size of the label is not enough.

Keyword Query

Semantics

Query Processing

Why do we need keyword search?

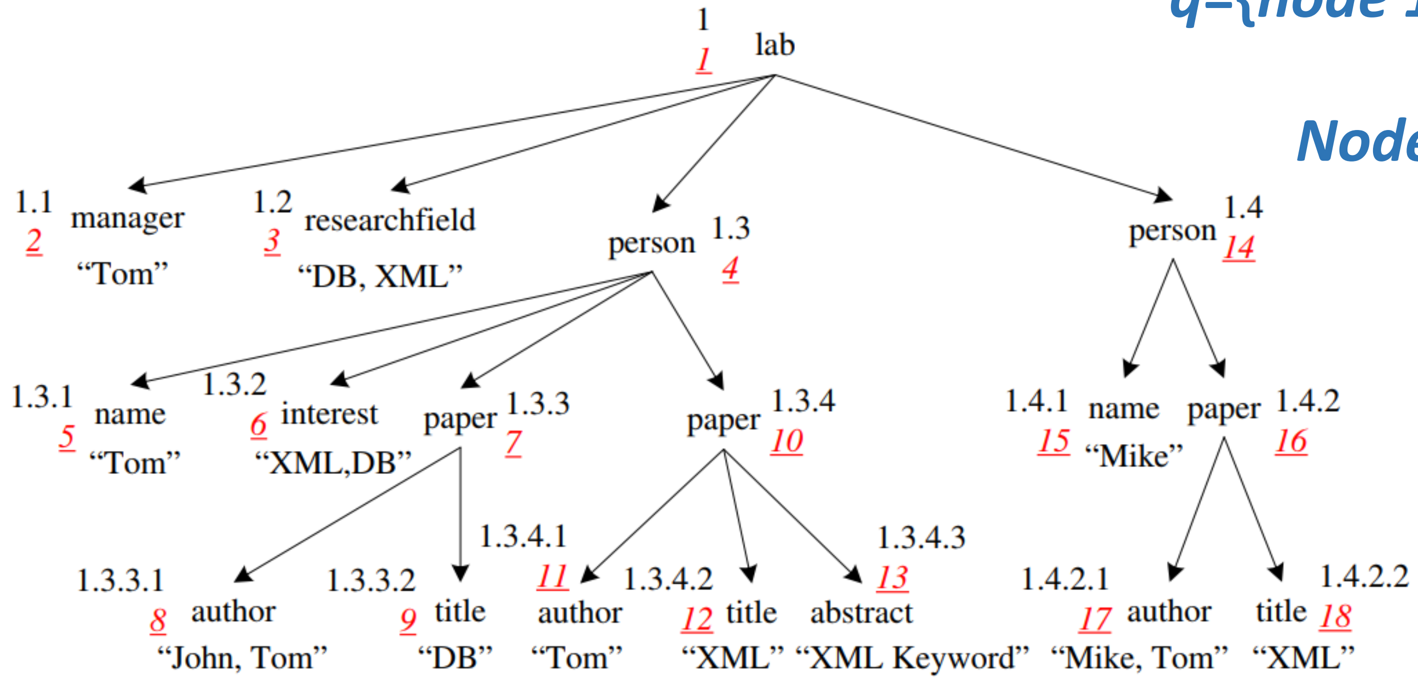
- Tree data (i.e., XML, JSON, *etc*) is becoming a **standard** in data representation
- Enables users to **access information** in tree databases
- Inspired by IR (Information Retrieval) style keyword search on the web, that is designed mostly for **text** databases
- Desirable to support keyword search in tree database *without the knowledge of* ***complex*** *query languages*
- The extreme success of web search engines makes keyword search the most popular search model for ordinary users

Common Ancestors

- Given two nodes in a tree
  - What are the common ancestors of these two nodes?
  - How to find them?
  - What's the difference between using Dewey and Interval labeling schemes?

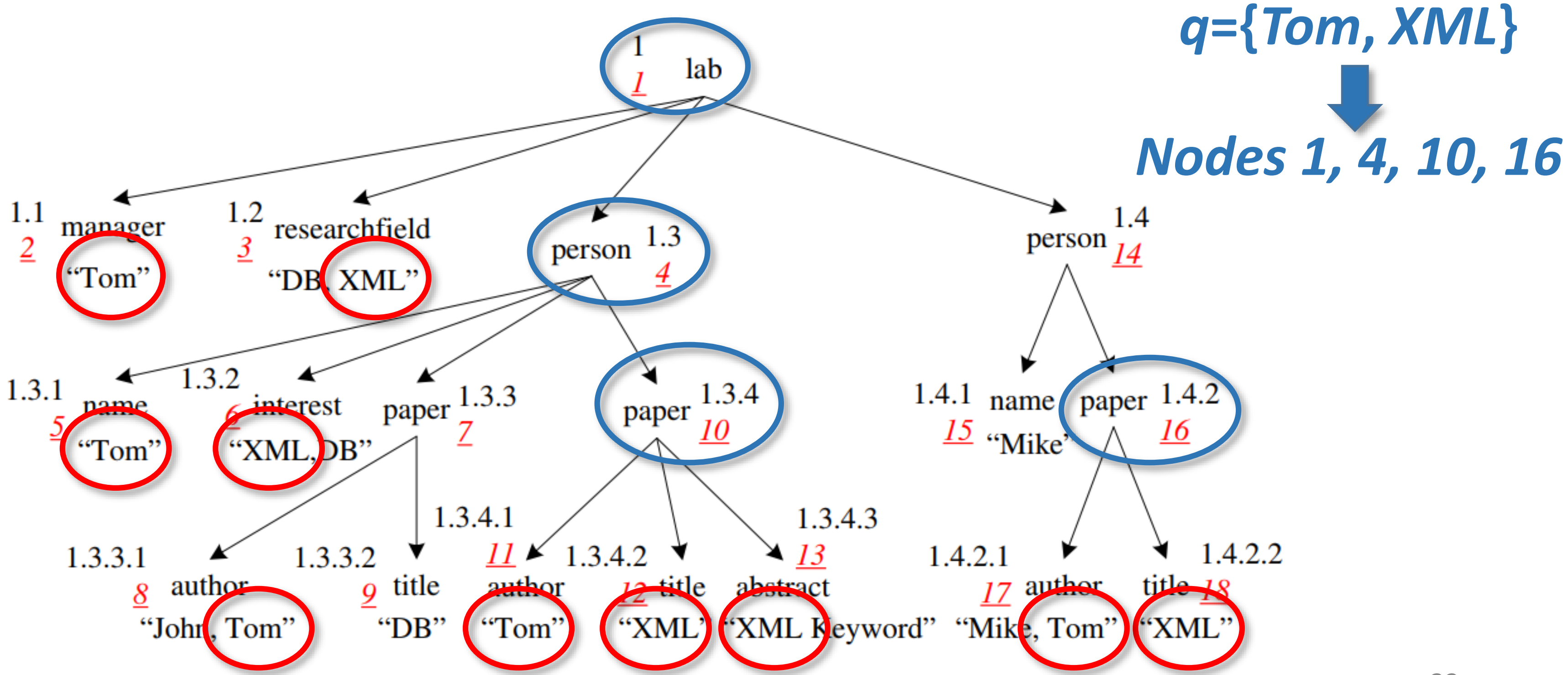
q={node 11, node 13}

Nodes 1, 4, 10



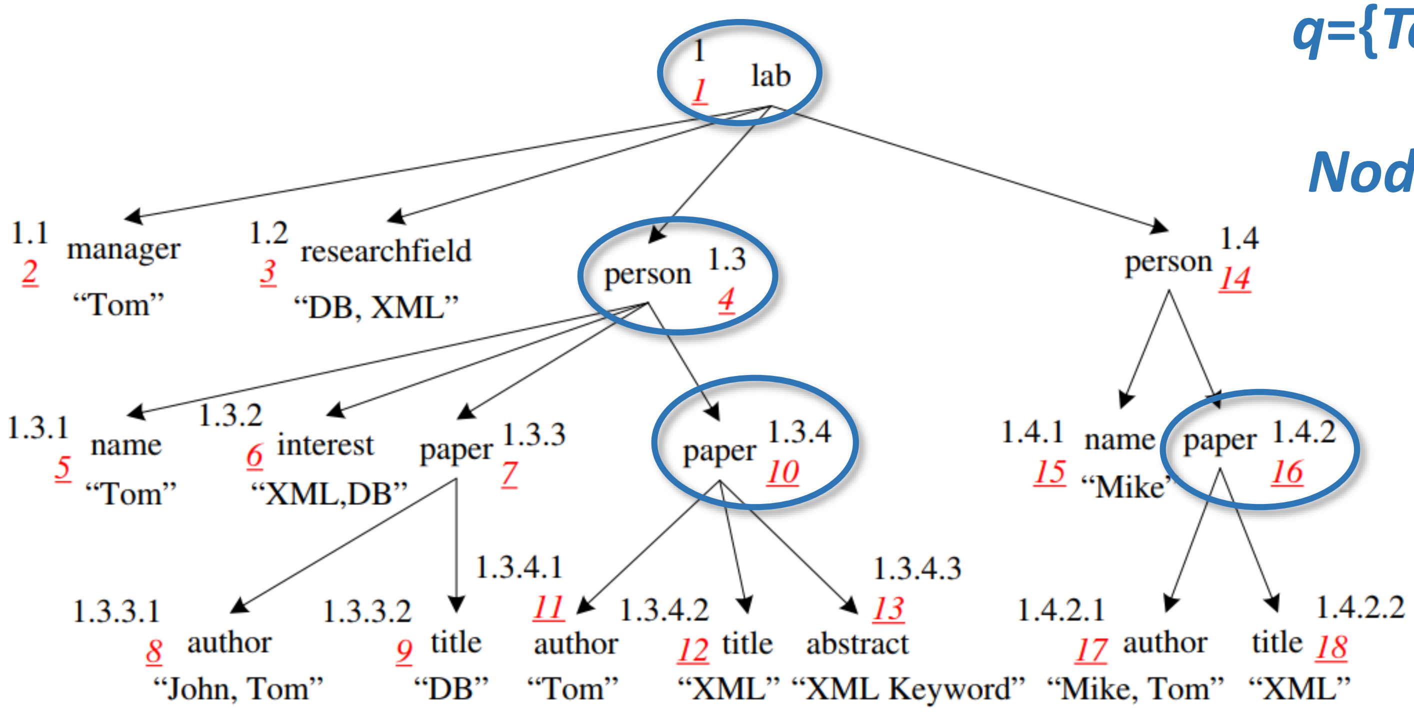
Semantic: LCA

- Lowest Common Ancestor
  - Given an XML document *d* and a subset of nodes v_1, v_2, ... ,v_m from d matching a set of query terms t_1,t_2, ...,t_n, the LCA of those nodes is a node *e* that is their common ancestor located farthest from the root of *d*.



Semantic: SLCA

- Smallest Lowest Common Ancestor:
  - Given a set of LCA nodes returned as the result of a query *q* on an XML document *d*, the corresponding SLCA nodes are the LCA nodes that contain no other LCA node as descendant.

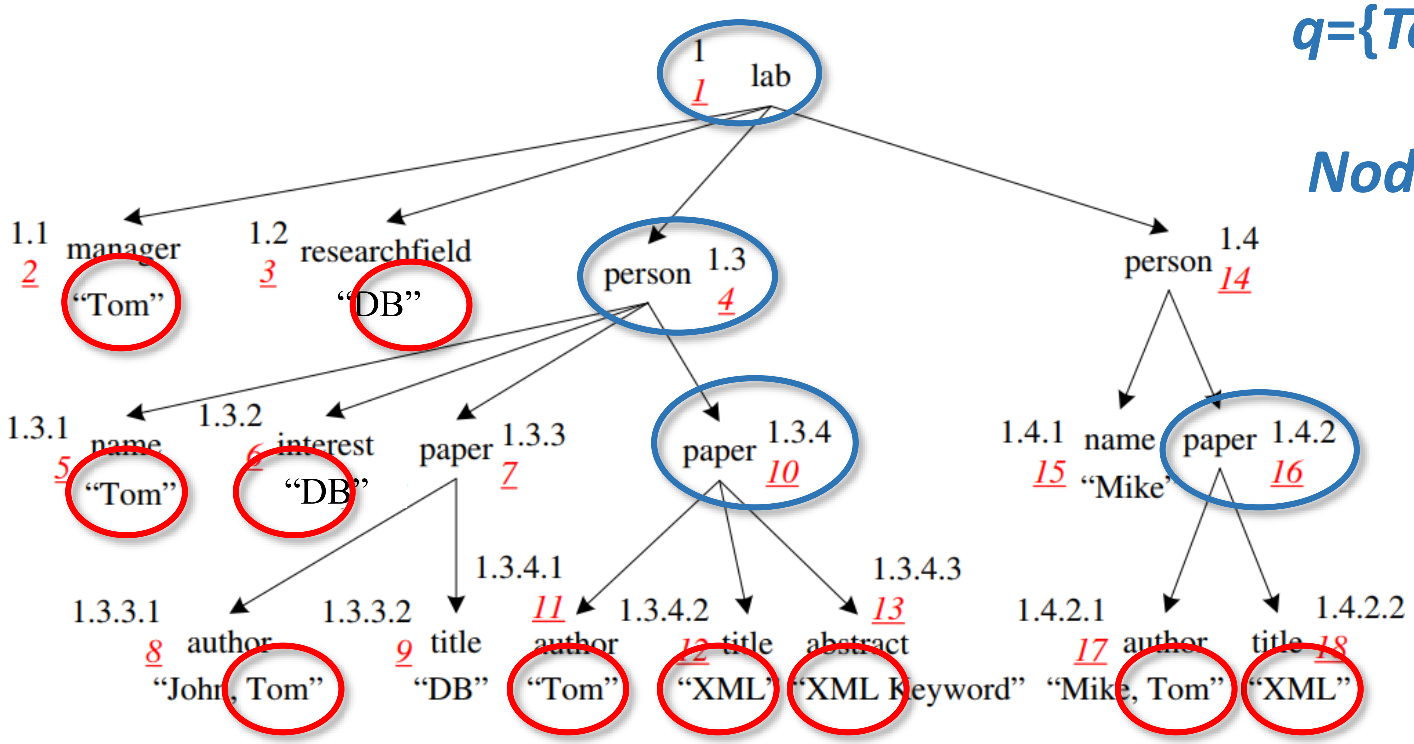


q={Tom, XML}

Nodes 10, 16

Semantic: ELCA

- Exclusive Lowest Common Ancestor:
  - Given a set of LCA nodes returned as the result of a query q on an XML document d, the corresponding ELCA nodes are those LCA nodes that contain at least one occurrence of each query term, excluding all their descendent LCA subtrees that also match any of those terms.



q={Tom, XML}

Nodes 10, 16

Other LCA-based Semaantics

Semantics	Definition	Existing algorithms	Returned nodes in Example 1
LCA	An LCA is a lowest common ancestor of a combination of matching nodes, i.e., each keyword corresponds to at least one matching node in the combination	XRANK Sigmod 2003	{ &o1, &o2, &o3, &o4, &o5 }
ELCA (Exclusive LCA)	*An ELCA is an LCA of a combination of matching nodes *An ELCA has its own witnesses, i.e., it does not share its matching nodes with its descendant ELCA nodes	*Index Stack EDBT 2008 *Hash Count EDBT 2010 *Top-K ICDE 2010 *Set-intersection ICDE 2011	{ &o1, &o4, &o5 }
SLCA (Smallest LCA)	*An SLCA is an LCA of a combination of matching nodes *There is no LCA node as its descendant	*XKSearch Sigmod 2005 *Multiway-SLCA WWW 2007 *Top-K ICDE 2010 *Set-intersection ICDE 2011	{ &o4, &o5 }
VLCA (Valuable LCA)	*A VLCA is an LCA of a combination of matching nodes *For each pair of matching nodes, all nodes in the path connecting them are of different types.	*XSEarch VLDB 2003 *VLCA CIKM 2007 *RLCA ADC 2010	{ &o2, &o3, &o4, &o5 }
MLCA (Meaningful LCA)	*An MLCA is an LCA of a combination of matching nodes *The LCA of matching nodes in the data tree belongs to the LCA of their node types in the schema tree	*MLCA VLDB 2004	{ &o1, &o2, &o3, &o4, &o5 }

Keyword Query

Semantics

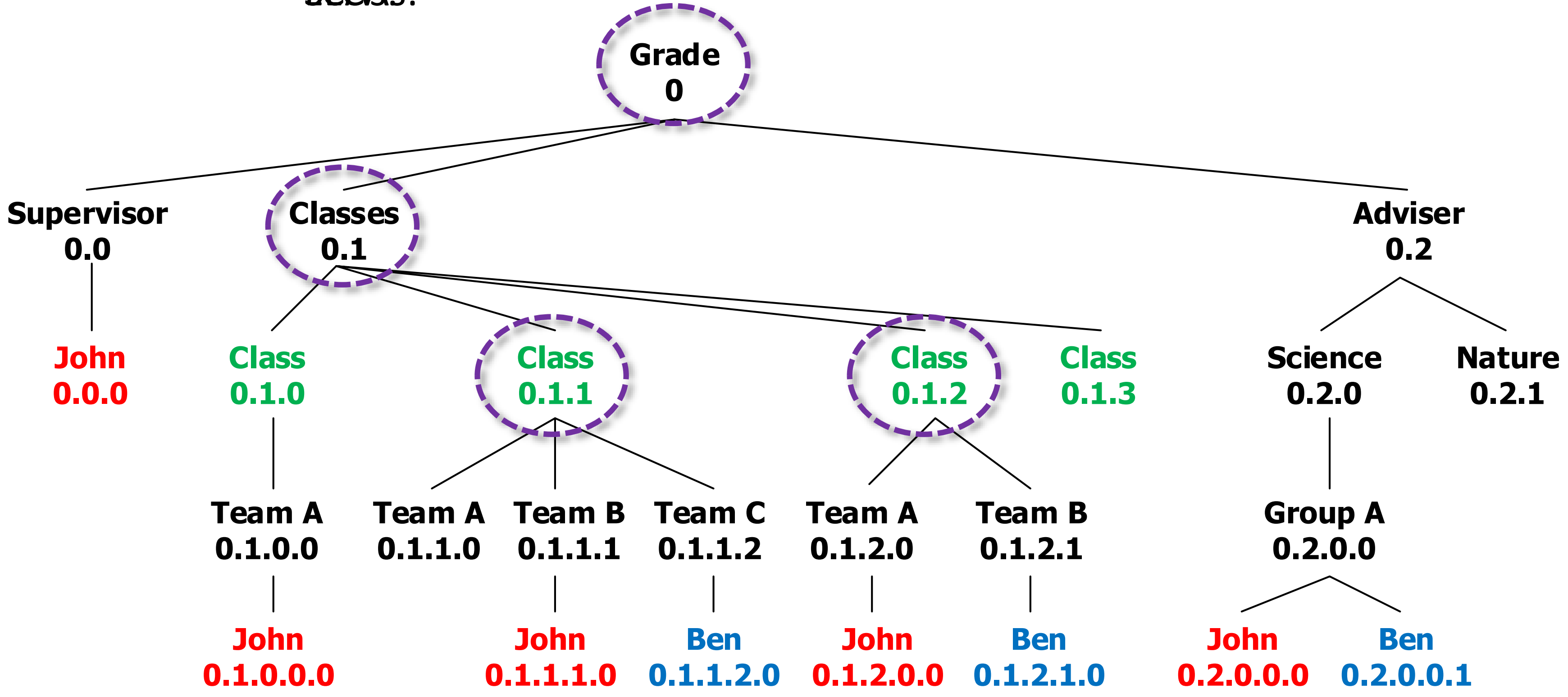
Query Processing

Find SLCAs

$Q = \{John, Ben, Class\}$

LCAs:

SLCAs:



Stack Algorithm for SLCA

$Q = \{John, Ben, Class\}$

$$John: \begin{Bmatrix} 0.0.0, \\ 0.1.0.0.0, \\ 0.1.1.1.0, \\ 0.1.2.0.0, \\ 0.2.0.0.0 \end{Bmatrix}$$

$$Ben: \begin{Bmatrix} 0.1.1.2.0, \\ 0.1.2.1.0, \\ 0.2.0.0.1 \end{Bmatrix}$$

$$Class: \begin{Bmatrix} 0.1.0, \\ 0.1.1, \\ 0.1.2, \\ 0.1.3 \end{Bmatrix}$$

	J	B	C
0	T	F	F
0	F	F	F
0	F	F	F

(a) node 0.0.0

0	F	F	T
1	F	F	F
0	T	F	F

(b) node 0.1.0

0	T	F	F
0	F	F	F
0	F	F	T
1	F	F	F
0	T	F	F

(c) node 0.1.0.0.0

1	F	F	T
1	T	F	T
0	T	F	T

(d) node 0.1.1

0	T	F	F
1	F	F	F
1	F	F	T
1	T	F	T
0	T	F	T

(e) node 0.1.1.1.0

0	F	T	F
2	F	F	F
1	T	F	T
1	T	F	T
0	T	F	T

(f) node 0.1.1.2.0

2	F	F	T
1	F	F	F
0	F	F	F

(g) node 0.1.2 report 0.1.1 as a SLCA

Distributed Algorithms

Reference UNPUBLISHED

End of Chapter 8