XML Technologies

XML Databases: Encodings

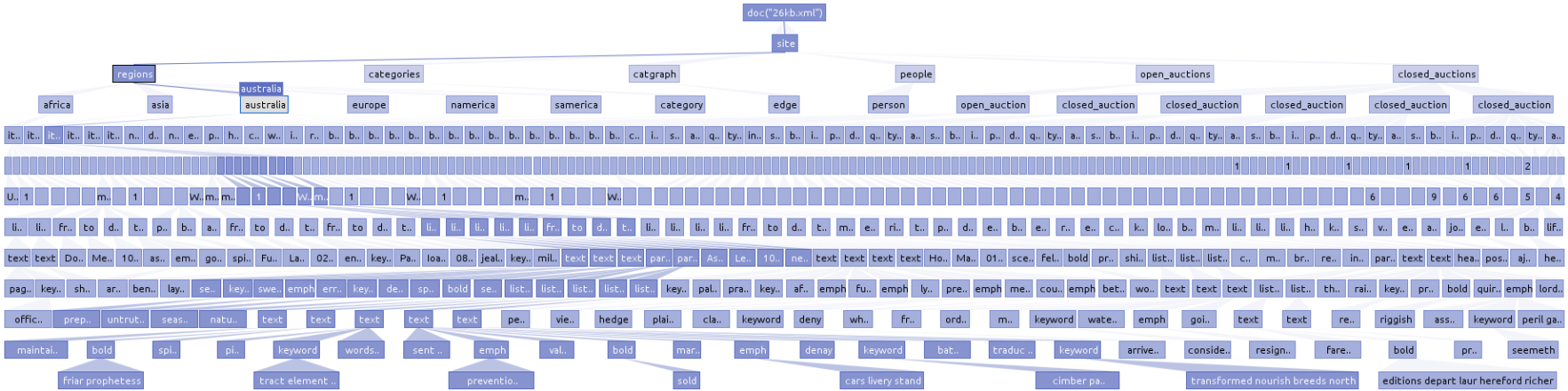
Christian Grün
Database & Information Systems Group
Universität Konstanz

Introduction

In this and the next meetings, we will...

- trace back the *roots* of today's XML databases
- analyze various representations (*encodings*) of XML documents
- look at how XML is broken down to single *bits and bytes*
- find out how data can be further *compressed*
- build up *indexes* for the resulting data structures

XML Structure



XML documents are nothing else than trees!

- more formally: *connected acyclic graphs*
- tree algorithms can be used for *processing* the document structure

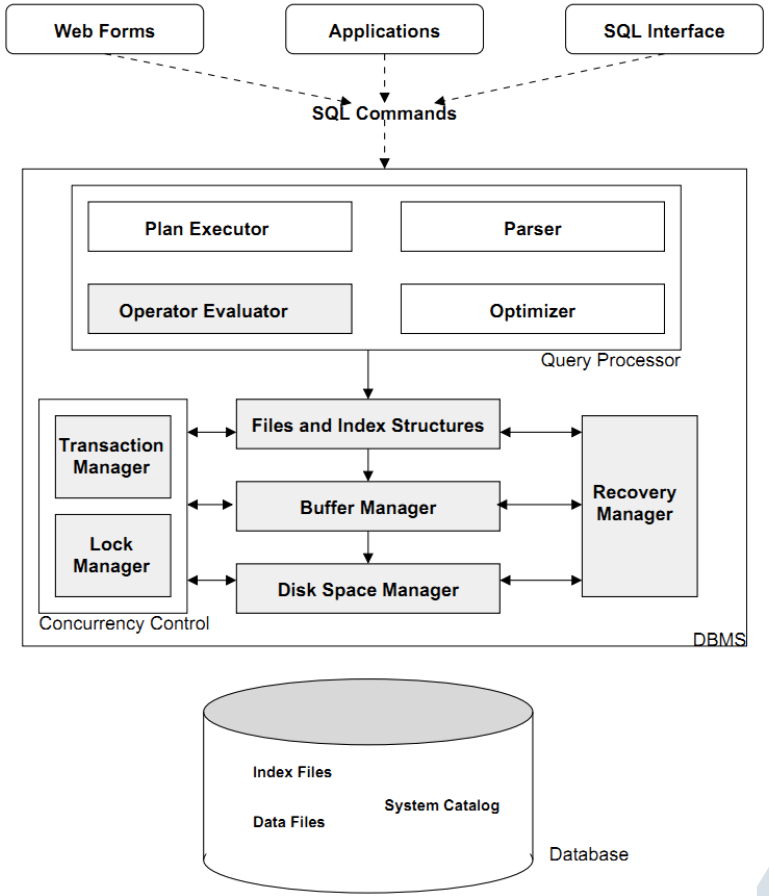
➘ can directed or cyclic graphs be represented as well?

DB Architecture

Which DBMS components do exist?

- interfaces (APIs)
- query processor:
  parsing, compilation, evaluation
- buffer management, caching
- transaction & locking
- recovery management
- data, indexes, meta information

How do XML databases look like?

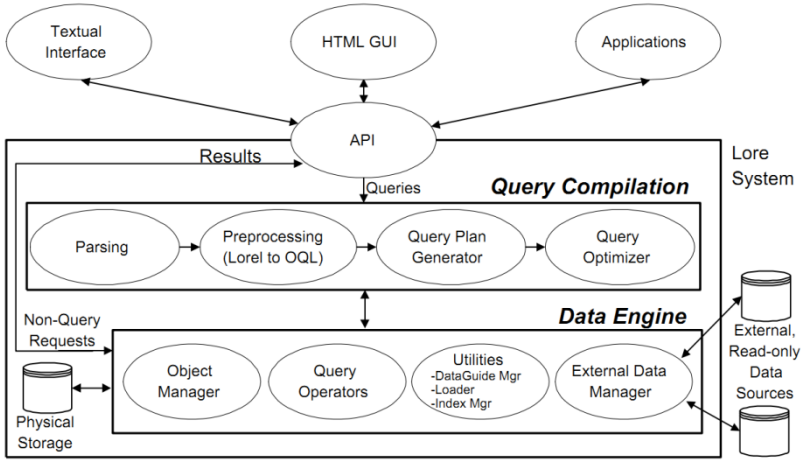


DB Architecture: LORE

Lightweight Object REpository

- developed at Stanford University
- based on *object-oriented* databases
- uses *OEM* (Object Exchange Model; looks similar to XML)
- novelty: database schema (*DataGuide*) is derived from input data
- novelty (at that time): query processor works ➘ schema-oblivious

McHugh et al. [1997]




```
<DB:: Eats { // A Sample Database
  <Restaurant {
    <Name "Darbar">
    <Entree {
      <Name str "Masala Dosa">
    }>
    <"Credit Card" "Visa">
  }>
}>
```

DB Architecture: STORED, NATIX

STORED: Semistructured To RElational

Mixed schema, derived from OEM input

- regular structures are stored relationally
- irregular structures are stored as graphs

Deutsch et al. [1999]

Taxpayer

oid	name	audit1	audit2	taxamount	taxevasion
o24	Gluschko	10/12/63		12332	
o21	Kosberg	11/1/68	10/12/77	0	likely
o20	Korolev	10/12/86		0	likely

Address1

taxpayeroid	street	no.	apt	zip
o24	Tyuratam		2C	07099
o21	Tyuratam	206		92443

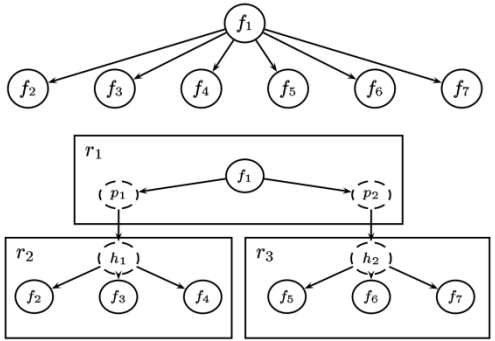
Address2

taxpayeroid	address
o20	Baikonur

NATIX

- maps XML documents to physical storage
- distributes XML snippets across disks blocks

Kanne & Moerkotte [2000]



DB Architecture

XML databases today:

- result of ten years of development
- well-established concepts

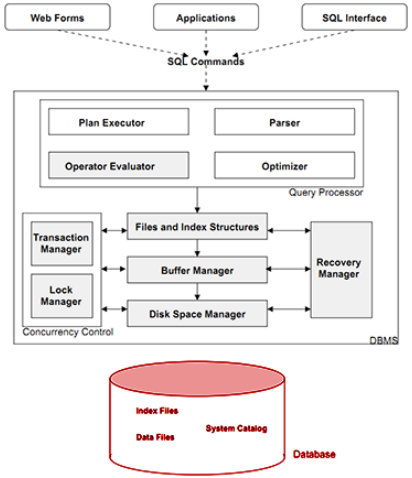
1. XML-enabled databases

- relational DBMS that *also* support XML

2. Native XML DBMS

- offer *specially tailored* data structures
- optimized for X-languages (XPath, XQuery, ...)

XML Encodings



Desired encoding $\mathcal{E}$ must...

1. be able to map XML to a *database* and *back* to XML ($\mathcal{E}^{-1}$)
2. reflect *node order* (essential for semi-structured data!)
3. support efficient *traversal* of the tree structure

Alternative wordings: *create*, *build*, *shred*, ... documents

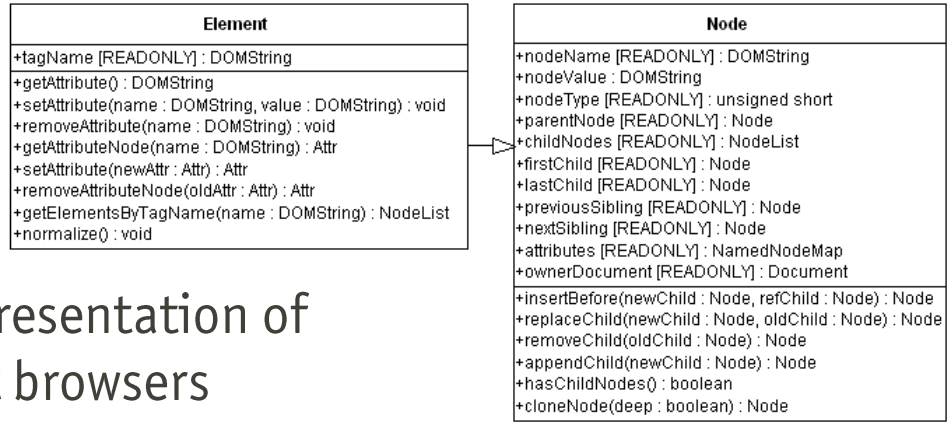
Three picked solutions

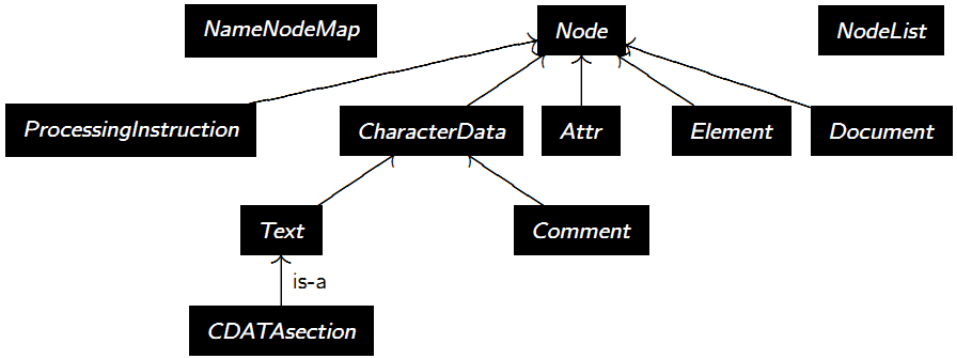
1. DOM: representation of XML nodes as *objects*
2. Hierarchic Labelings: Dewey, ORDPATH
3. Numbering Schemes: *pre*, *post*, *size*, *dist*, ...

XML Encodings: DOM

Document Object Model

- popular W3C standard
- first effort to standardize the representation of HTML/XML documents in internet browsers (Netscape, Internet Explorer)
- XML nodes are mapped to *main-memory* objects
- node objects are equipped with functions to query and modify the tree structure





XML Encodings: DOM

DOM Example in Java

```
// create document instance
DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
DocumentBuilder db = dbf.newDocumentBuilder();
Document doc = db.parse("addressbook.xml");

// query document nodes
NodeList nl = doc.getElementsByTagName("address");
int l = nl.getLength();
System.out.println("Result: " + l);
```

✎ How could this be expressed in XPath?

Drawbacks

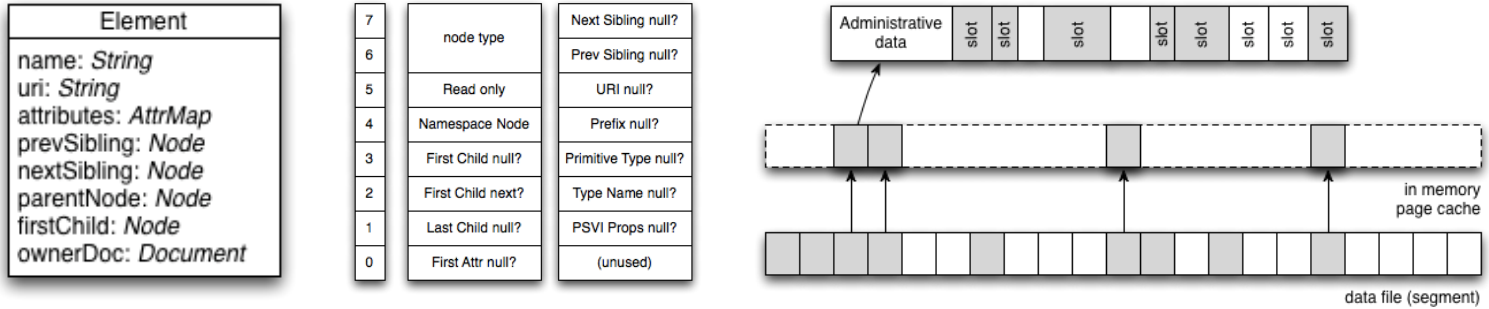
- occupies *lots of space*
- operates in *main-memory*

XML Encodings: DOM

Persistent DOM

- binary, persistent representation of DOM objects
- node objects and references to related objects are mapped to disk

Edwards & Hope [2000]: Persistent DOM – An architecture for XML repositories
Lv et al. [2002]: Performance evaluation of a DOM-based XML database



Probst [2010]: Processing Arbitrarily Large XML using a Persistent DOM

XML Encodings: Hierarchic Labelings

Dewey Decimal Classification (DCC)

- attempt to organize all human knowledge in a *hierarchical scheme*
- used worldwide in *library catalogs*
- model can be arbitrarily *extended*

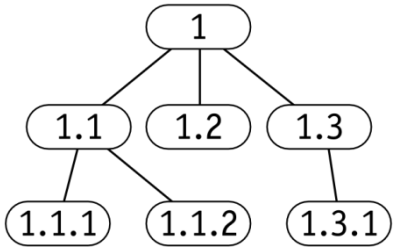
The Lancashire cotton industry : a study in economic development

Assigned DDC Code: 338.476772109427 6

3 Economics, Education, Society
33 Economics and Management
338 Industries, Products
338.1 – 338.4 Specific kinds of industries
338.4 Secondary Industries and Services
338.47 Goods and Services
338.471 – 338.479 Subdivisions for Goods and Services
338.476 Technology
338.4767 Manufacturing
338.47677 Textiles
338.476772 Textiles of Seed hair fibres
338.4767721 Cotton
338.47677210 Facet Indicator for Standard Subdivision
338.476772109 Historical, geographic, persons treatment
338.4767721094 Europe Western Europe
338.47677210942 England and Wales
338.476772109427 Northwestern England and Isle of Man
338.4767721094276 Lancashire

Dewey IDs

- general concept of assigning hierarchical numbers to *descendant tree nodes*
- drawback: updates are expensive (✎ why?)



XML Encodings: Hierarchic Labelings

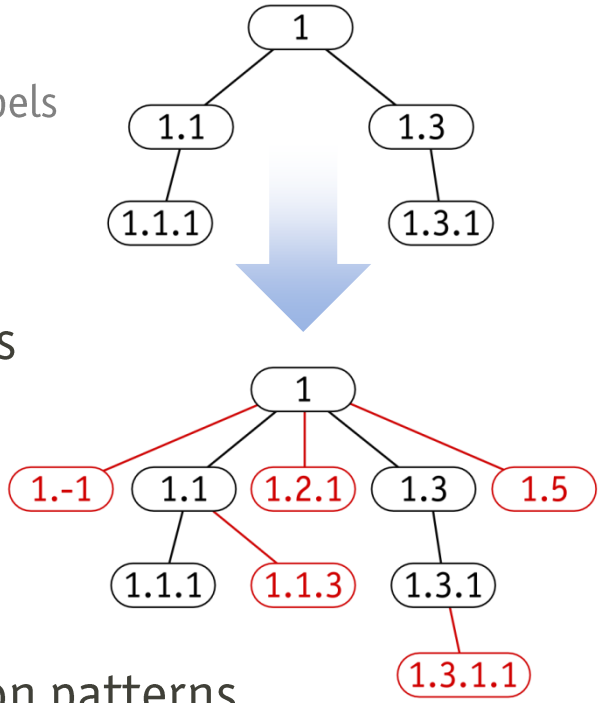
ORDPATH

O'Neil et al. [2004]: ORDPATHs: Insert-Friendly XML Node Labels

- *odd numbers* are assigned to initial labels
- even numbers are used as *insertion markers*

☞ no renumbering required when inserting nodes

✎ insert node between (1.2.1) and (1.3)!



Drawbacks

- hierarchical labels have *variable length*
- structure might *degenerate* for certain insertion patterns

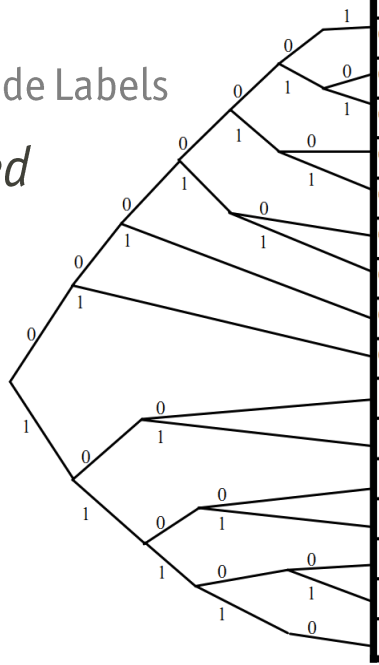
XML Encodings: Hierarchic Labelings

ORDPATH: Compression

O'Neil et al. [2004]: ORDPATHs: Insert-Friendly XML Node Labels

- hierarchical labels are *bit-wise compressed* into a fixed-size representation
- compressed bit format:
  $N_0 . N_1 \ldots . N_k \dashrightarrow L_0 O_0 L_1 O_1 \ldots L_k O_k$
  - L_i: bitstring (encoded bit length of O_i)
  - O_i: bits of value N_i

What is the bit pattern for the ORDPATH label `1.8.33`?



Bitstring	L_i	O_i value range
0000001	48	[-2.8x10^{14},-4.3x10^{9}]
0000010	32	[-4.3x10^{9}, -69977]
0000011	16	[-69976, -4441]
000010	12	[-4440, -345]
000011	8	[-344, -89]
00010	6	[-88, -25]
00011	4	[-24, -9]
001	3	[-8, -1]
01	3	[0, 7]
100	4	[8, 23]
101	6	[24, 87]
1100	8	[88, 343]
1101	12	[344, 4439]
11100	16	[4440, 69975]
11101	32	[69976, 4.3x10^{9}]
11110	48	[4.3x10^{9}, 2.8x10^{14}]

XML Encodings: Hierarchic Labelings

ORDPATH: Compression

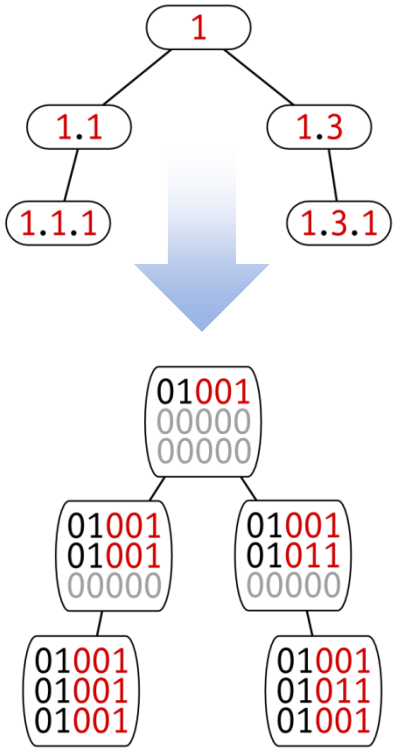
O'Neil et al. [2004]: ORDPATHs: Insert-Friendly XML Node Labels

- the resulting bit sequences preserve *node order* and contain *ancestry relationships*
- m appears before n if its value is smaller:

  (m = 1.1.1 = 010010100101001 = 9513) <
  (n = 1.3 = 010010101100000 = 9568)
- m is *ancestor* of n if m starts with substring of n:

  m = 1 : 01001 is a substring of
  n = 1.3 : 0100101011

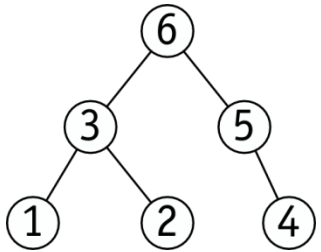
Drawback: deeply nested nodes lead to *overflow*



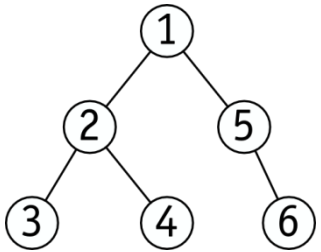
XML Encodings: Numbering Schemes

Preorder, postorder, inorder

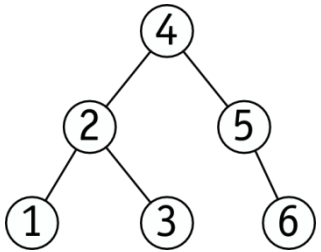
Knuth [1968]: The Art of Computer Programming, Volume 1



traverse left subtree
traverse right subtree
visit the root



visit the root
traverse left subtree
traverse right subtree



traverse left subtree
visit the root
traverse right subtree

✎ Which ordering is represented by which tree?

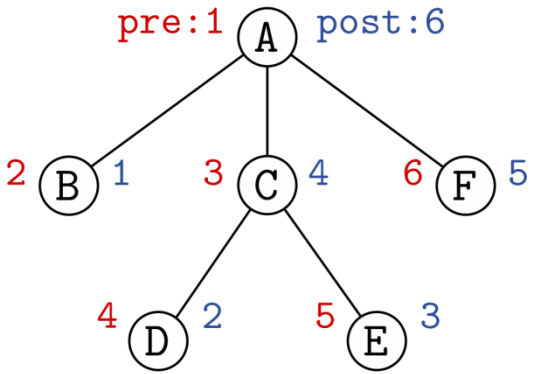
☞ As all numbers are unique, they may also serve as node IDs

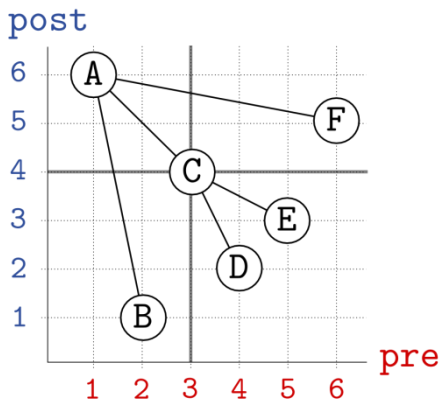
XML Encodings: Numbering Schemes

Pre/post encoding

Grust [2002]: Accelerating XPath Location Steps

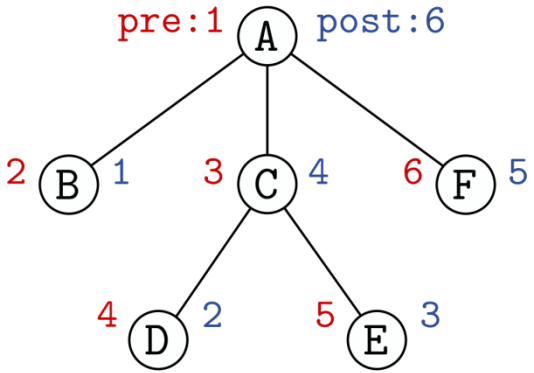
```
<A>
  <B/>
  <C>
    <D/><E/>
  </C>
  <F/>
</A>
```





- all pre and post values can be mapped to a *two-dimensional plane*
- each node partitions the plane into *four regions*

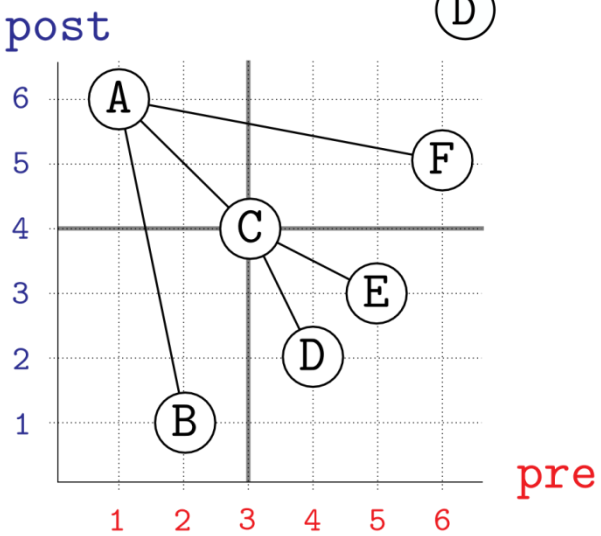
XML Encodings: Numbering Schemes



Pre/post encoding

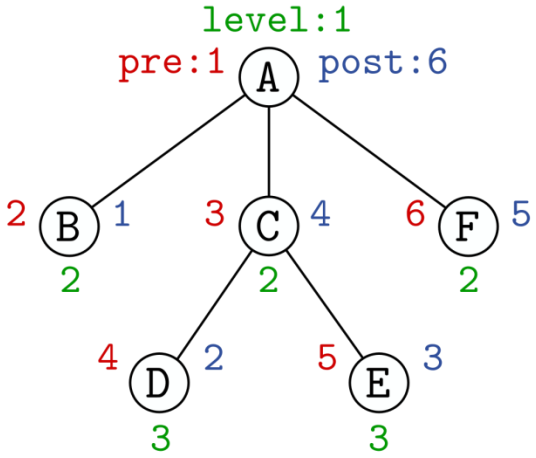
Grust [2002]: Accelerating XPath Location Steps

- `n'` is an *ancestor* of `n` if:
  pre(n') < pre(n) ∧ post(n') > post(n)
- `n'` is a *descendant* of `n` if:
  pre(n') > pre(n) ∧ post(n') < post(n)
- `n'` is a *following* node of `n` if:
  pre(n') > pre(n) ∧ post(n') > post(n)
- `n'` is a *preceding* node of `n` if:
  pre(n') < pre(n) ∧ post(n') < post(n)



How can you find child nodes?

XML Encodings: Numbering Schemes

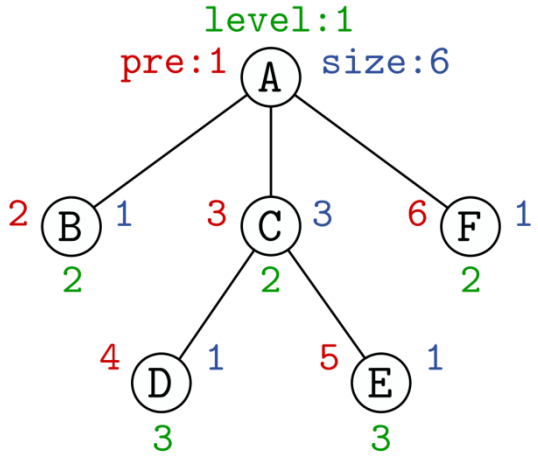


Pre/post/level encoding

- `n'` is *parent* of `n` if:
  pre(n') < pre(n) ∧ post(n') > post(n) ∧ level(n') = level(n) – 1
- `n'` is a *child* of `n` if:
  pre(n') > pre(n) ∧ post(n') < post(n) ∧ level(n') = level(n) + 1
- `n'` is a *following sibling* of `n` if it has `p` as parent and if:
  pre(n') > pre(n) ∧ post(n') > post(n) ∧ post(n') < post(p) ∧ level(n') = level(n)
- `n'` is a *preceding sibling* of `n` if it has `p` as parent and if:
  pre(n') < pre(n) ∧ post(n') < post(n) ∧ pre(n') > pre(p) ∧ level(n') = level(n)

✎ Can *following sibling* nodes be found without reference to parent node?

XML Encodings: Numbering Schemes



Pre/size/level encoding

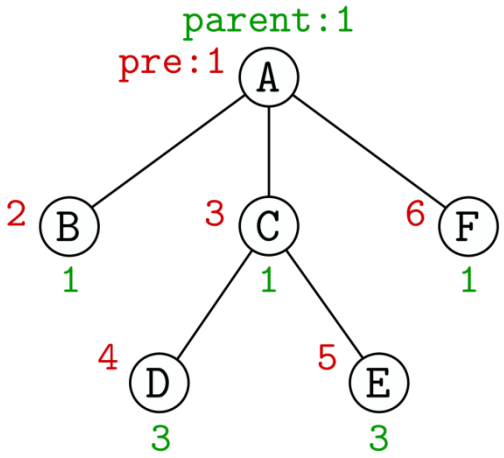
Inspired by Li & Moon [2002]: Indexing and Querying XML Data...

The size value represents no. of *descendants* + 1.
It simplifies numerous axis traversals:

- `n'` is a *descendant* of `n` if:
  $pre(n) < pre(n') < pre(n) + size(n)$
- `n'` is a *following sibling* of `n` if:
  $pre(n') = pre(n) + size(n) \wedge level(n') = level(n)$
- `n'` is a *preceding sibling* of `n` if:
  $pre(n') = pre(n) - size(n') \wedge level(n') = level(n)$

Important *relationship*: $size(n) = post(n) - pre(n) + level(n)$

XML Encodings: Numbering Schemes



Pre/parent encoding

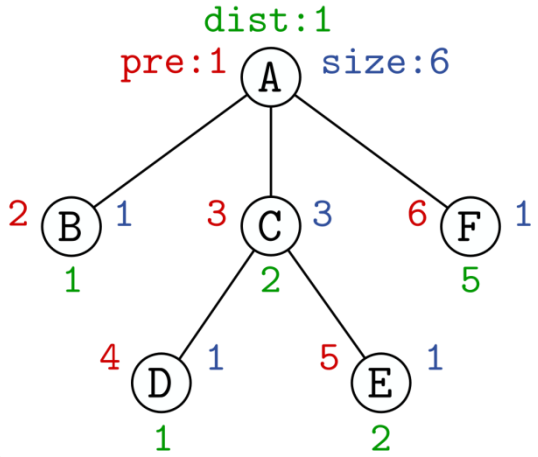
Grün et al. [2006]: Pushing XPath Accelerator to its Limits

- the parent value represents a direct mapping between *children* and their *parents*:
  `n'` is a *parent* of `n` if pre(n') = parent(n)

✎ Exercise: `n'` is a *following sibling* of `n` if...

- the pre/parent pair constitutes a minimal encoding to reconstruct the original document (no post, size or level needed)
- most update operations based on pre/parent are expensive (✎ why?)

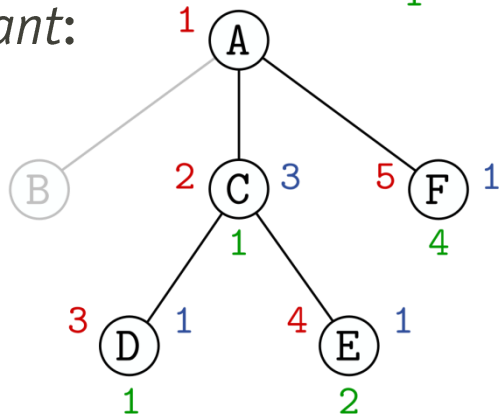
XML Encodings: Numbering Schemes



Pre/dist/size encoding

Grün et al. [2009]: XQuery Full Text Implementation in BaseX

- the dist value is the *distance* to the parent node:
  `n'` is a *parent* of `n` if pre(n') = pre(n) – dist(n)
- in contrast to parent, dist is *update-invariant*: subtrees preserve the *original distance* value when moved or inserted
- the size property has been retained, as it speeds up some axis operations (see previous slides)



XML Encodings: Numbering Schemes: Updates