Informatics 1: Data & Analysis

Lecture 9: Trees and XML

Ian Stark

School of Informatics
The University of Edinburgh

Tuesday 12 February 2013
Semester 2 Week 5

Smart Data Hack

Make cool apps
Learn new stuff
Eat free food
Win cool prizes
(like Nexus 7s and Kindle Fires)

Starts next Monday (18th)
REGISTER NOW!
data.inf.ed.ac.uk/ilwhack

Lecture Plan

XML

We start with technologies for modelling and querying *semistructured data*.

- Semistructured Data: Trees and XML
- Schemas for structuring XML
- Navigating and querying XML with XPath

Corpora

One particular kind of semistructured data is large bodies of written or spoken text: each one a *corpus*, plural *corpora*.

- Corpora: What they are and how to build them
- Applications: corpus analysis and data extraction

Lecture Plan

XML

We start with technologies for modelling and querying *semistructured data*.

- Semistructured Data: Trees and XML
- Schemas for structuring XML
- Navigating and querying XML with XPath

Corpora

One particular kind of semistructured data is large bodies of written or spoken text: each one a *corpus*, plural *corpora*.

- Corpora: What they are and how to build them
- Applications: corpus analysis and data extraction

Reading Around the Subject

For a very brief summary and sales pitch, read this short introduction:

World Wide Web Consortium (W3C).
XML Essentials
http://www.w3.org/standards/xml/core, W3C 2010.

Ramakrishnan and Gehrke's database book briefly introduces XML in §7.4:

R. Ramakrishnan and J. Gehrke.
Database Management Systems.
McGraw-Hill, third edition, 2003.

For a more comprehensive introduction, see Chapter 2 of:

A. Møller and M. I. Schwartzbach.
An Introduction to XML and Web Technologies.
Addison-Wesley, 2006.

There are multiple copies in the Main Library HUB section.

There's More to Life than Structured Data

Relational databases record data in tables conforming to fixed schemas, satisfying various constraints about uniqueness and cross-referencing.

That can usefully capture real-world constraints in a way which supports automatic validation and efficient querying.

However it can also be helpful in some situations to structure data in a less rigid way. For example:

- When the data has no strong inherent structure; or there is structure, but it varies from item to item;
- When we wish to *mark up* (annotate) existing unstructured data (say, English text) with additional information (e.g. linguistic structure, or meaning);
- When the structure of the data changes over time, perhaps as more data accumulates.

Trees

Often this kind of semistructured data is modelled using *trees*.

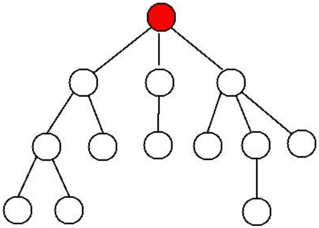
These are mathematical trees, not vegetation. You can recognise them by the fact that they grow branches downwards from a root at the top.

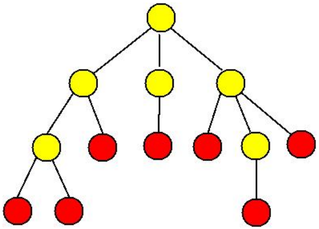
Nature notes

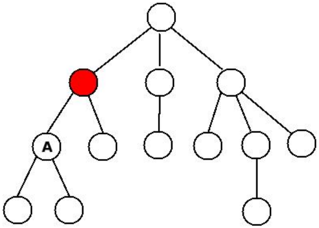
- A tree is a set of linked nodes, with a single root node.
- Each node is linked to a set of zero or more children, also nodes in the tree.
- Every node has exactly one parent, except for the root node which has none.
- A node with no children is a leaf; other nodes are internal.
- Two nodes with a common parent are sibling nodes.

Trees contain no loops, and from each node there is always exactly one route back up to the root.

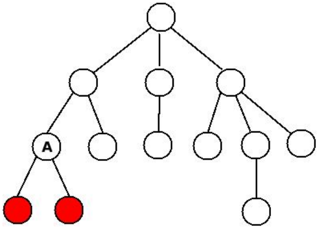
Know Your Trees


Root node


Leaves and internal nodes



Parent of A



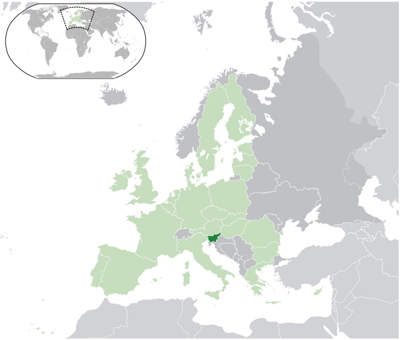
Children of A

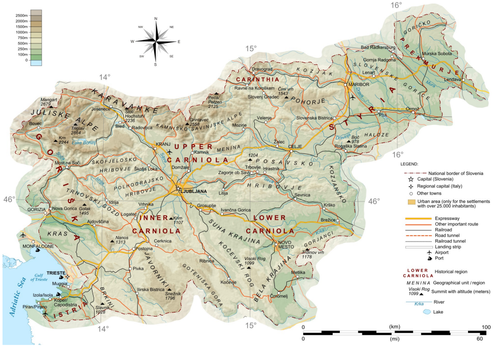
Semistructured Data Models

There are several tree-like data models used with semistructured data.

We shall work with the XPath data model, developed for semistructured data represented using XML (of which more shortly).

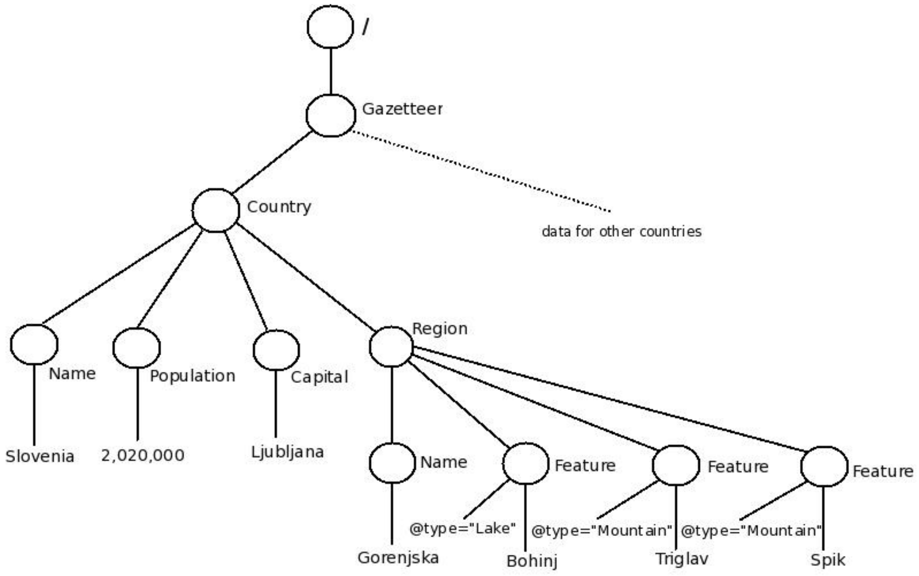
The next slide shows an example of data structured according to the XPath data model, with a fragment of a geographical directory or *gazetteer*.





Wikipedia: NuclearVacuum

Sample Semistructured Data



XPath Node Types

Root Node: This is the root of the tree, labelled / .

Element Nodes: These are labelled with *element names*, categorising the data below them. In this example the element names are: Gazetteer, Country, Name, Population, Capital, Region, and Feature.

In the XPath data model, internal nodes other than the root are always element nodes.

The root node must have exactly one element node as child, called the *root element*. Here the root element is Gazetteer.

Text Nodes: Leaves of the tree storing textual information. In this example there are text nodes with strings "Slovenia", "2,020,000", "Ljubljana", "Gorenjska", "Triglav", "Bohinj" and "Spik".

Attribute Nodes: . . .

More XPath Node Types

Attribute Nodes: Leaves of the tree assigning a value to some *attribute* of an element node.

In the example, we use the @ symbol to identify attributes. In this case we see a single attribute type, associated with the Feature element, which is assigned the text values "Lake" and "Mountain".

In the XPath data model, attribute nodes are treated differently from other node types. For example, although the parent of an attribute node is an element node, when we talk about the children of this parent node we generally don't include the attribute nodes.

One aim of the XPath model, and the XML language, is that data should be *self-describing*: lots of the data in these trees is there to give information about the meaning of other data.

People argue about this

Understanding an XPath Data Tree

In a tree like this, the meaning of data at a text node depends on all the element nodes that appear along the path from the root of the tree to the text node, and on the values of their associated attributes.

We usually write these paths with a / separator, beginning at the root. For example, the path to the text node containing "Bohinj" is

/Gazetteer/Country/Region/Feature/

and the value of the type attribute of the associated Feature element is "Lake". This tells us that Bohinj is a feature in a region in a country in the gazetteer, and that the type of feature is a lake.

Note that to get further information (such as the name of the country, Slovenia), we would need to follow another path from the relevant ancestor element (in this case, the Country element).

Understanding an XPath Data Tree

In a similar way the meaning of an element node depends on the path to that node from the root of the tree.

For example, in the gazetteer a Name element node is used in two different ways:

- A path /Gazetteer/Country/Name/ leads to a text node containing the name of a country.
- A path /Gazetteer/Country/Region/Name/ leads to a text node containing the name of a region.

All of this structure in an XPath data tree can be written out in plain text using the *Extensible Markup Language* XML.

XML: Extensible Markup Language

XML is formal language for presenting the kind of semistructured data we have just seen. It is a *markup* language in that it provides a way to *mark up* ordinary text with additional information.

XML was developed in the 1990's building on the *Standard General Markup Language* SGML and the *Hypertext Markup Language* HTML. It aimed to be simpler than SGML, but more general than HTML.

XML has a simple text-based format which is suitable for machine-to-machine communication, by automatically generating and parsing data files, as well as being moderately human-readable.

(Compare, for example, Abstract Syntax Notation One)

XML has become the standard mechanism for publishing data on the web.

There's a lot of it about

The "extensible" part of XML means it can be applied to all kinds of semistructured data, with customised versions for any number of application domains. For example, all of the following are based on XML:

- XHTML for web pages
- SVG for scalable vector graphics
- OOXML for Microsoft Office documents .docx, .pptx, .xlsx
- GLADE for GTK+ user interface descriptions
- GML, the Geography Markup Language
- MusicXML for musical scores
- FpML, the Financial Products Markup Language

Homework: Find an SVG file, and look at its XML content.
Advanced: Find a .docx file, and look at its XML content.

Sample Semistructured Data in XML

```
<Gazetteer>
    <Country>
        <Name>Slovenia</Name>
        <Population>2,020,000</Population>
        <Capital>Ljubljana</Capital>
        <Region>
            <Name>Gorenjska</Name>
            <Feature type="Lake">Bohinj</Feature>
            <Feature type="Mountain">Triglav</Feature>
            <Feature type="Mountain">Spik</Feature>
        </Region>
    </Country>
    <!-- data for other countries here -->
</Gazetteer>
```

XML Elements

The building blocks of XML documents are *elements*, also called *tags*.

The content of a thing element is marked with a *start tag* <thing> at the beginning and an *end tag* </thing> at the end.

Elements must be properly nested. For example:

```
<Country><Region> ... </Region></Country>
```

is acceptable XML, whereas

```
<Country><Region> ... </Country></Region>
```

is not.

Elements in XML are case sensitive, so <REGION> is different from <Region> and <region>.

Content of XML Elements

Each element in an XML document has *content*:

- The content of the Capital element

  <Capital>Ljubljana</Capital>

  is the text string "Ljubljana".
- The Region element above has as content one Name element together with three Feature elements.
- The root element Gazetteer has the whole document as content.

An element may possibly have empty content: <thing></thing>.
This can be abbreviated by the single hybrid tag: <thing/>.

Attributes for XML Elements

Any element can have descriptive attributes which provide additional information about the element. For example:

```
<Feature type="Mountain"> ... </Feature>
```

declares that the attribute type of the given Feature element has value Mountain.

Attribute values are always enclosed in either single or double quotation marks.

A single element may have multiple different attributes, each with its own value declared in the element start tag:

```
<thing attr1="value1" attr2="value2" ... > ... </thing>
```

In designing an XML representation for semistructured data, there is sometimes a tension between putting information in the content of an element, or as one of its attributes.

Matching XML with the Tree Model

Every XML document naturally represents a tree structure in the XPath data model:

- Each XML element corresponds to an element node of the tree.
- The XML root element corresponds to the root element of the tree (the one below the root node).
- The text content of an individual XML element corresponds to a child text node of the corresponding element node in the tree.
- An attribute definition in an element's start tag corresponds to a child attribute node of the corresponding element node in the tree.

Of course, this correspondence is there because the data model was designed that way.

Other XMLicities

XML files can contain comments `<!-- almost anything here -->`. The full XPath data model also has comment nodes.

Well-formed XML documents should all begin with a declaration something like

```
<?xml version="1.0" encoding="UTF-8"?>
```

Both XML and the data model allow for all kinds of *processing instruction nodes* also written `<?...?>`.

Because XML documents are plain text files, there are some unexpected consequences in the tree structure:

- Order of children matters
- Whitespace sometimes matters *but in ways too horrible to describe.*