

# Informatics 1: Data & Analysis

## Lecture 9: Trees and XML

Ian Stark

School of Informatics  
The University of Edinburgh

Tuesday 12 February 2019  
Semester 2 Week 5

This is Inf1-DA Lecture 9, in Week 5.

Next week is the **Festival of Creative Learning** (FCL). All lectures, tutorials, labs and coursework are suspended and replaced by a series of alternative events across the University.

<http://www.festivalofcreativelearning.ed.ac.uk>

Check the FCL calendar and sign up now: some activities run all week, some are one-off.

The week after that, from Monday 26 February, is Teaching Week 6.

*Some courses may have not noticed that this is the week numbering scheme.*

# Sample FCL Events — Book Today

!

## Internet of Things Challenge

*Wednesday–Friday*

Team event run by the Informatics Embedded and Robotics Society (EaRS)

## Reading the Newspaper like a Mathematician

*Friday*

Analysing validity of arguments; understanding statistics; spotting fallacies.

## iGEM Sandpit: Synthetic Biology for Social Challenges

*Monday/Wednesday/Friday*

Introduction to the international student contest for Genetically Engineering Machines.

## Mathematics Escape Room

*Friday*

30 minutes to escape with the proof to Fermat's Last Theorem. (Waiting list)

## Digital Concrete

*Wednesday–Friday*

3D modelling software, CNC milling machines, and moulding concrete.

## 3D Structures with Bioplastic

*Monday & Thursday*

Make-your-own bioplastics; moulding; digital fabrication.

## XML — The Extensible Markup Language

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

# Outline


- 1 Introduction
- 2 The XPath Data Model
- 3 XML

# XML: 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.

For a more comprehensive introduction, see Chapter 2 of this:

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

The **Main Library HUB** has multiple copies. The Library also offers the following online:

 **E. T. Ray.**  
*Learning XML.*  
Second edition, O'Reilly 2003.      <http://edin.ac/1bOkwui>

# 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 may be helpful in some situations to arrange data in a less rigid way. For example:

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

# 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.

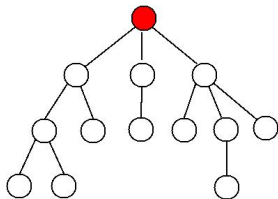
## Nature notes

- 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.

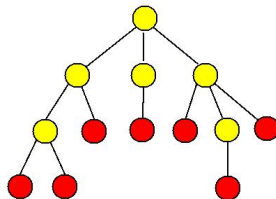
Trees have no loops, and from each node there is exactly one route back up to the root.



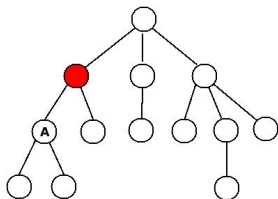
# Know Your Trees



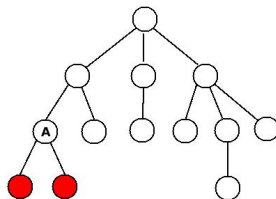
Root node



Leaves and internal nodes



Parent of A



Children of A

# Outline

1 Introduction

2 The XPath Data Model

3 XML

# 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.

The next slide shows an example of data structured according to the XPath data model, in the form of a geographical factbook.

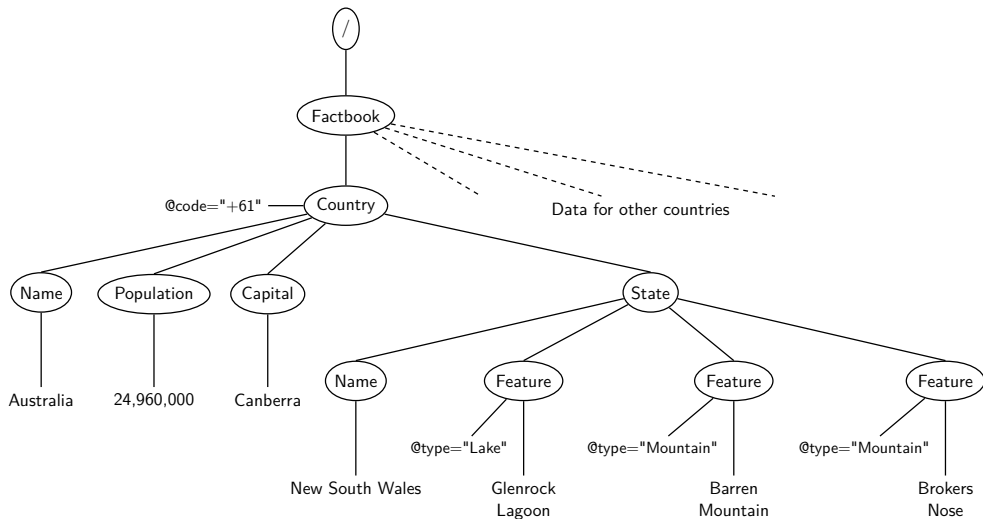


Wikimedia Commons: Ssolbergj



Wikimedia Commons: TUBS

# 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: `Factbook`, `Country`, `Name`, `Population`, `Capital`, `State`, 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 `Factbook`.

**Text Nodes:** Leaves of the tree storing textual information. In this example there are text nodes with strings `"Australia"`, `"24,960,000"`, `"Canberra"`, `"New South Wales"`, `"Glenrock Lagoon"`, `"Barren Mountain"` and `"Brokers Nose"`.

**Attribute Nodes:** ... (see next slide) ...

# More XPath Node Types

**Attribute Nodes:** Leaves 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 an attribute **code** for **Country**, which gives the international telephone prefix; and another attribute **type** associated with the **Feature** element, which is assigned text values like **"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 "Glenrock Lagoon" is

`/Factbook/Country/State/Feature/`

and the value of the `type` attribute of the associated `Feature` element is "Lake". This tells us that Glenrock Lagoon is a feature in a state in a country in the factbook, and that the type of feature is a lake.

Note that to get further information (such as the name of the country, Australia), we would need to follow a different path from an ancestor element (in this case, the `Country` element).

# Understanding an XPath Data Tree

In the same way the meaning of an *element node* depends on the path to that node from the root.

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

- A path `/Factbook/Country/Name/` leads to a text node with the name of a country.
- A path `/Factbook/Country/State/Name/` leads to a text node with the name of a state.

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



# Outline

- 1 Introduction
- 2 The XPath Data Model
- 3 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.

Like SQL, XML has a textual format which is suitable for robust machine-to-machine communication, by automatically generating and parsing data files, as well as being moderately human-readable.

(Compare, for example, the binary encoding in *Abstract Syntax Notation One*)

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

As a **human-readable data serialization** language it does, however, have competitors such as JSON or YAML.

# 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
- MathML for writing mathematics
- GLADE for GTK+ user interface descriptions
- GML, the Geography Markup Language
- MusicXML for musical scores
- FpML, the Financial Products Markup Language

## Read This

- *XML Essentials* from the W3C <http://www.w3.org/standards/xml/core>
- Sections 2.1–2.5 of Møller and Schwartzbach; download PDF from Learn or read the book at the Main Library.

## Do This

- Find an SVG file and open it in a text editor to study its XML content.
- Find a .docx file, and look at its XML content.

This format is in fact a zipped archive of XML files, so you will need to unzip it first. Depending on your platform, this may require renaming the .docx extension as .zip



# Travelling Salesman



Given a table of travel times between all  $n^2$  pairs of towns:

- What is the quickest route to visit them all?
- Is there any route shorter than X?

Checking any route is fast.

However, there a lot of routes to check ( $n!$ ).

Can we do it faster?

Not much, no. The fastest algorithm known takes time related to  $2^n$ .

This **exponential growth** means that a small problem can quickly become very large.



# Travelling Salesman



This **travelling salesman problem** is more widely applicable than the name suggests, and it also arises in:

- Commercial distribution logistics;
- Optimising chip layout;
- Assembling DNA sequence fragments; *etc.*



The travelling salesman problem is **NP-hard**: one of a large class of equivalent computational challenges where:

- Checking a potential solution is quick. . .
- . . . but there are a lot of potential solutions. . .
- . . . and we don't know how to do it any faster.

There's a bounty on these. You find a fast way to solve travelling salesman, or a proof that there isn't one, and you collect \$1M.

Apply to the *Clay Mathematics Institute*, 10 Memorial Boulevard, Providence, Rhode Island, USA.







P.S. If you are an actual salesman:

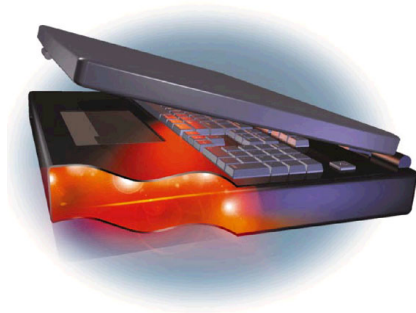
- Distances based on real space simplify the general problem;
- There are fast algorithms which with very high probability return an answer very close to the optimum.

Other NP-hard problems are not be so easy to work around.

Various physical factors set upper bounds on how much computation is possible:

- The speed of light,  $c$ ;
- Planck's constant,  $\hbar$ ;
- Universal gravitational constant,  $G$ ;
- Quantisation of energy states;
- Heisenberg uncertainty in observation;
- Mass/energy equivalence.

For a 1kg computer, this sets a limit on computation of  $10^{50}$  operations per second on  $10^{31}$  bits.



Working at  $10^9\text{K}$ , “the ultimate laptop looks like a small piece of the big bang”.

[Seth Lloyd, *Ultimate Physical Limits to Computation*, Nature 406:1045–154, August 2000]

Matter organised to provide the greatest possible computing power is fancifully known as **computronium**.

In the 1960's Hans-Joachim Bremermann was one of the first people to estimate upper limits to computation.

The *Bremermann limit* is the computation which could be performed using the earth, over the period of its existence so far.

This is around  $10^{93}$  bits of computation.

That's enough to solve the travelling salesman problem for 300 cities.

But just the once.



# Sample Semistructured Data in XML

```
<Factbook>
  <Country code="+61">
    <Name>Australia</Name>
    <Population>24,360,000</Population>
    <Capital>Canberra</Capital>
    <State>
      <Name>New South Wales</Name>
      <Feature type="Lake">Glenrock Lagoon</Feature>
      <Feature type="Mountain">Barren Mountain</Feature>
      <Feature type="Mountain">Brokers Nose</Feature>
    </State>
  </Country>
  <!-- data for other countries here -->
</Factbook>
```

# 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><State> ... </State></Country>
```

is acceptable XML, whereas

```
<Country><State> ... </Country></State>
```

is not.

Elements in XML are case sensitive, so `<STATE>` is different from `<State>` and `<state>`.

# Content of XML Elements

Each element in an XML document has *content*:

- The content of the **Capital** element

`<Capital>Canberra</Capital>`

is the text string "**Canberra**".

- The **State** element given earlier has as content one **Name** element together with three **Feature** elements.
- The root element **Factbook** 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 must be enclosed in 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 such *processing instruction nodes*, all written in the form `<?...?>`.

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

- Order of children matters (unless they are attributes)
- Whitespace sometimes matters *but in ways too horrible to describe*.

# Two Examples of XML

## MusicXML

<http://www.musicxml.com/tutorial/hello-world>

## Financial products Markup Language FpML

<http://www.fpml.org>      <https://is.gd/fpml511ex>

# Summary

## Semistructured Data

Not all data fits the rectangular tables of relational databases. Some data may have less structure; or the structure varies from item to item; or the structure changes over time. Often this is set out as a *tree*.

## The XPath Data Model

The XPath data model describes a specific kind of tree structure: starting at a *root node* then branching through named *element nodes* with *attribute nodes* off to the side and *text nodes* at the leaves.

## XML: The Extensible Markup Language

XML gives a textual form for XPath data trees that allows robust communication of semistructured data. It also serves as a *markup* language for annotating plain text with structural information.