Part II — Semistructured Data

XML:

II.1 Semistructured data, XPath and XML

II.2 Structuring XML

II.3 Navigating XML using XPath

Corpora:

II.4 Introduction to corpora

II.5 Querying a corpus

Recommended reading:

§§3.1–3.4 of [XWT]

pp. 948–949 of [DMS] (superficial coverage only)

On-line XPath tutorial: **`http://www.w3schools.com/xpath/`**

How do we extract data from an XML document?

Since an XML document is a text document, one option is to use methods based on text search.

But this ignores the element structure of the document.

A better alternative is to use a dedicated language for forming queries based on the *tree structure* of an XML document

This has many uses, for example:

- Performing database-style queries directly on data published as XML
- Extracting annotated content from marked-up text documents
- Identifying information captured in the tree structure itself

XQuery and XPath

XQuery is a powerful declarative query language for extracting information from XML documents.

However, the XQuery language is too complex for this course. (See [XWT] for further information.)

XPath is a sublanguage of XQuery, used specifically for navigating XML documents using *path expressions*.

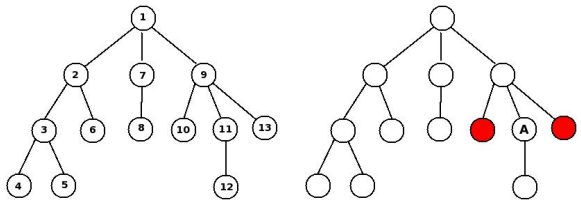
XPath can be viewed as a rudimentary query language in its own right.

It is also an important component of many XML application languages other than XQuery (e.g., XML Schema, XSLT, XLink, XPointer).

Location paths

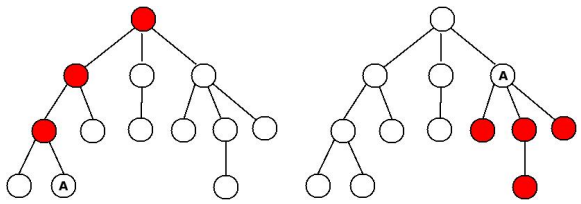
A *location path* (a.k.a. *path expression*) retrieves a *set* of nodes from an XML document tree.

- The location path describes a set of possible paths from the root of the tree.
- The set of nodes retrieved is the set of all nodes reached as final destinations of the described paths.
- This set of nodes is returned as a list of nodes (without duplicates) sorted in *document order* (the order in which the nodes appear in the XML document)



Document order

Siblings of A



Ancestors of A

Descendants of A

Example location paths

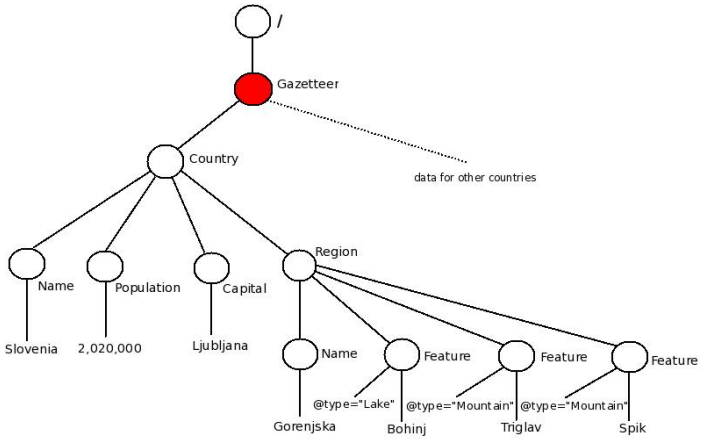
The next few slides illustrate a selection of location paths. Each is given twice: above using the full XPath syntax, and below using a convenient abbreviated syntax.

In each case, the retrieved nodes are highlighted in red. These nodes will be returned as a list in document order.

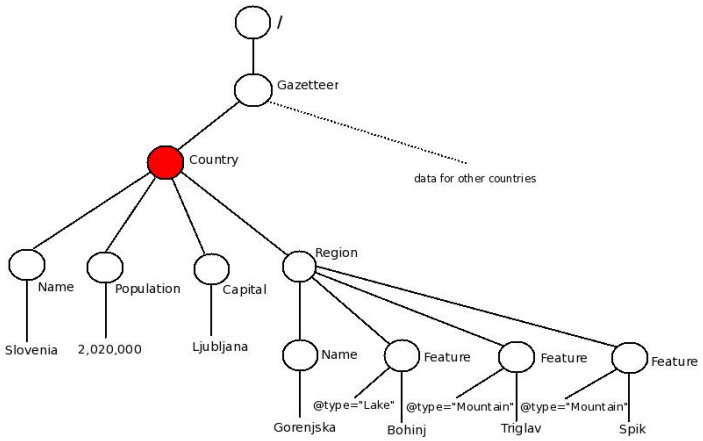
Paths are built up step-by-step as the location path is read from left-to-right.

Each path is constructed by a *context node* that travels over the tree, according to certain rules, depending on the continuation of the location path expression.

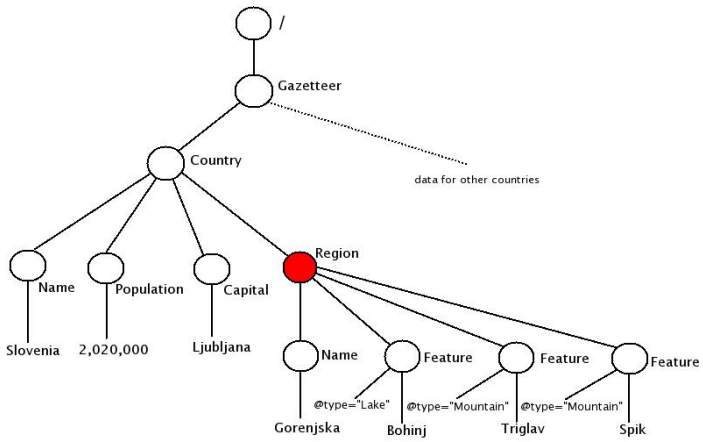
The slash **/** at the start of a location path indicates that the starting position for the context node is the root node.



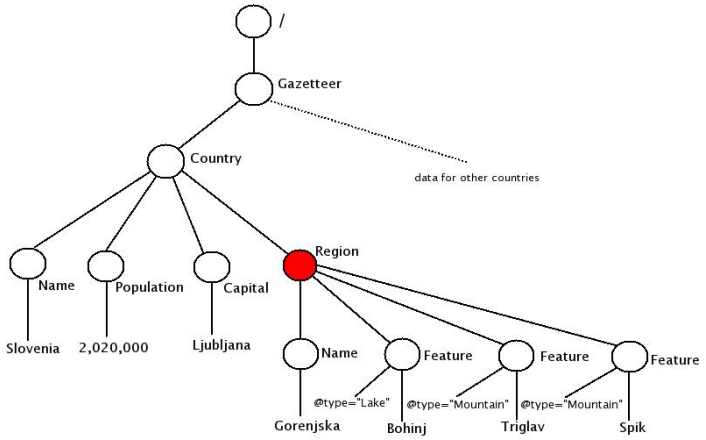
/child::Gazetteer
/Gazetteer



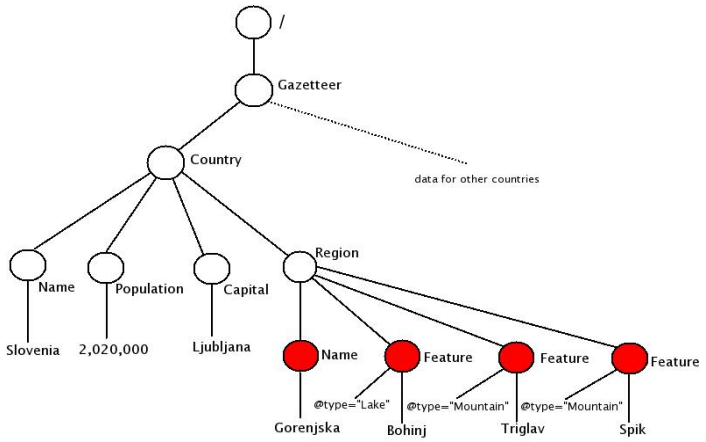
/child::Gazetteer/child::Country
/Gazetteer/Country



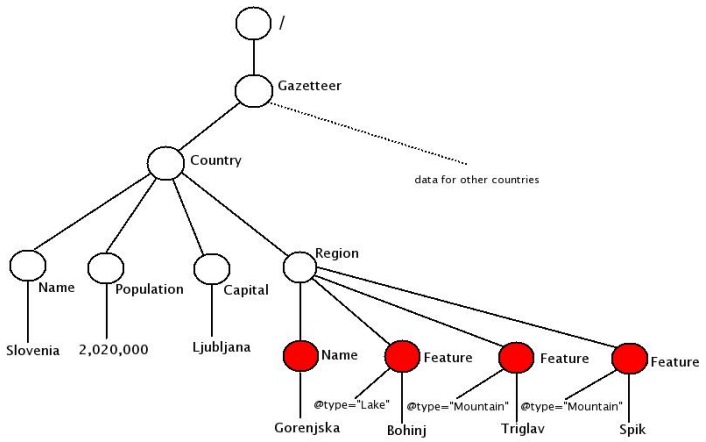
/child::Gazetteer/child::Country/child::Region
/Gazetteer/Country/Region



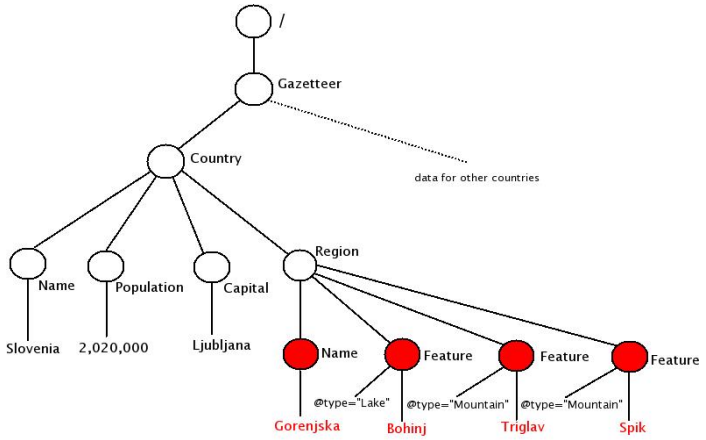

```
/descendant::Region
//Region
```

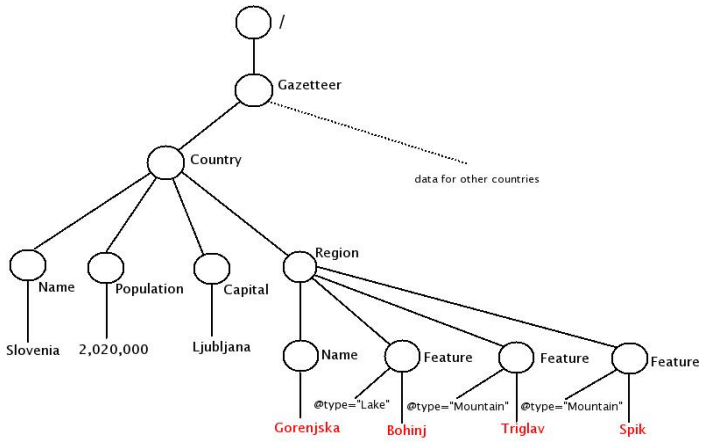

```
/descendant::Region/child::*
//Region/*
```

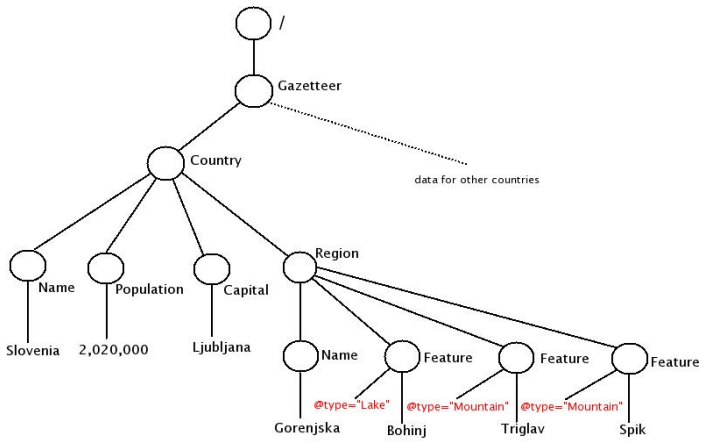

```
/descendant::Region/descendant::*
//Region//*
```




```
/descendant::Region/descendant::node()
//Region//node()
```


```
/descendant::Region/descendant::text()
//Region//text()
```


```
/descendant::Feature/attribute::type
//Feature/@type
```

General unabbreviated syntax of location paths

A *location path* is a sequence of *location steps* separated by a **/** character.

A *location step* has the form

axis*::*nodeTest predicate*

- The *axis* tells the context node which way to move.
- The *node test* selects nodes of an appropriate type from the tree.
- The optional *predicates* supply conditions that need to be satisfied for the path to be allowed to count towards the result.

N.B., the previous examples contained only axes and node tests.

A selection of axes

- **child**: the children of the context node (remember, an attribute node does not count as a child node)
- **descendant**: the descendants of the context node (again, an attribute node does not count as a descendant).
- **parent**: the unique parent of the context node (where the context node must not be the root node).
- **attribute**: all attribute nodes of the context node (which must be an element node).
- **self**: the context node itself (this is useful in connection with abbreviations).
- **descendant-or-self**: the context node together with its descendants.

A selection of node tests

Node tests filter the nodes selected by the current axis according to the type of node.

- **text()**: selects only character data nodes.
- **node()**: selects all nodes.
- *****: if the axis is **attribute** then all attribute nodes are selected; for any other axis, all element nodes are selected.
- ***name***: selects the nodes with the given name.

  The names used for node tests in the earlier examples were: **Gazetteer**, **Country**, **Region**, **Feature** and **type**.

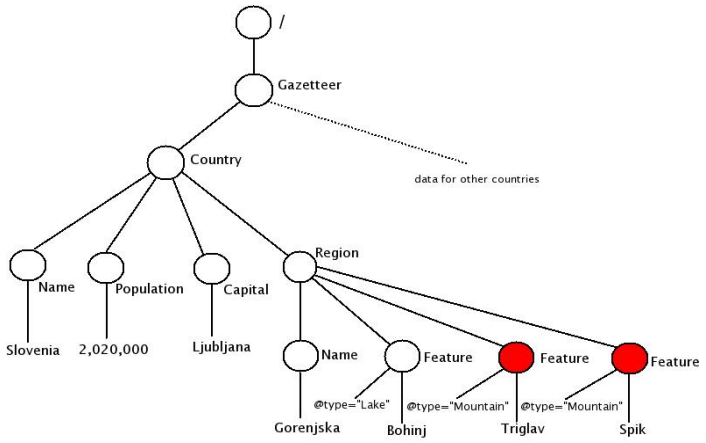
Predicates

The node test in a location step may be followed by zero, one or several *predicates* each given by an expression enclosed in square brackets.

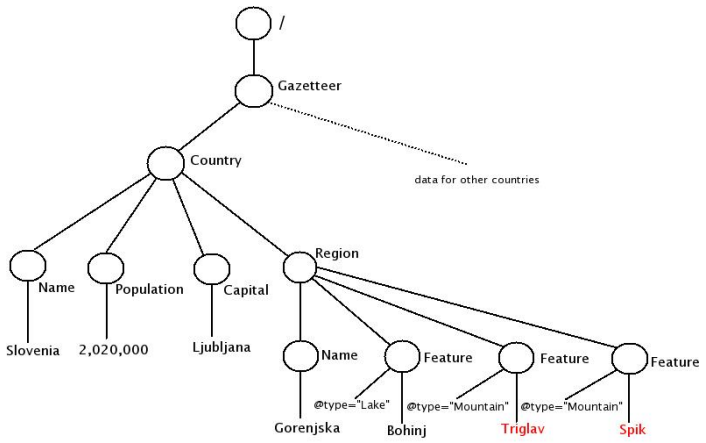
Common examples of predicates are:

- **`[ locationPath ]`**
  This selects only those nodes for which there exists a continuation path (from the current node) matching ***`locationPath`***.
- **`[ locationPath = value ]`**
  Selects those nodes for which there exists a continuation path matching ***`locationPath`*** such that the final node of the path is equal to ***`value`***.

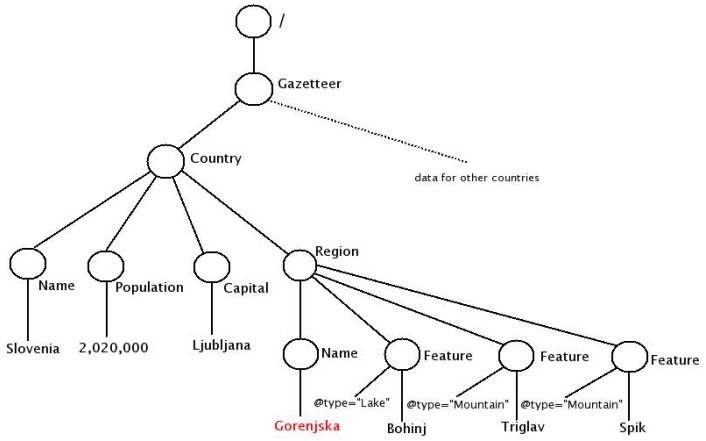
The full syntax of XPath predicate expressions is rather powerful, but beyond the scope of the course.




```
/descendant::Feature[attribute::type='Mountain']
//Feature[@type='Mountain']
```


```
/descendant::Feature[attribute::type='Mountain']/child::text()
//Feature[@type='Mountain']/text()
```




```
//Feature[@type='Mountain']/../Name/text()
```


XPath as a query language

The previous examples illustrate XPath as a rudimentary query language.

The queries formulated are:

- Slide II: 60 : Find every feature element for which the feature is a mountain.
- Slide II: 61 : Find the name of every mountain.
- Slide II: 62 : Find the name of every region in which there is a mountain.

The last query was given only in abbreviated form. The full version is more cumbersome:

```
/descendant::Feature[attribute::type='Mountain']/
  parent::*/child::Name/child::text()
```


Abbreviated syntax

The abbreviated syntax is more economical and often (but not always!) more intuitive.

The XPath abbreviations are:

- The syntax `child::` may be omitted from a location step altogether. (The child axis is chosen as default.)
- The syntax `@` is an abbreviation for: `attribute::`
- The syntax `//` is an abbreviation for:

  `/descendant-or-self::node()/`
- The syntax `..` is an abbreviation for: `parent::node()`
- The syntax `.` is an abbreviation for: `self::node()`

Queries and alternatives

Consider again the last query above:

Find the name of every region in which there is a mountain.

An alternative location path for this is:

```
//Region[Feature/@type='Mountain']/Name/text()
```

Similarly, consider:

Find the name of countries containing a feature called Everest.

Two queries for this are:

```
//Feature[text()='Everest']/../../Name/text()
//Country[.//Feature/text()='Everest']/Name/text()
```


One subtle point

A subtle point with XPath is illustrated by the second solution above to:

Find the name of countries containing a feature called Everest.

While the given query (repeated below) is correct,

```
//Country[.//Feature/text()='Everest']/Name/text()
```

the following (natural) attempt would be incorrect:

```
//Country[//Feature/text()='Everest']/Name/text()
```

The problem is that the location path `//Feature/text()` starts with a `/` character, and this means that XPath interprets this path as starting at the root node, whereas the path needs to start at the current node.

The omission of a necessary '`.`' character at the start of a predicate expression is a common source of errors in XPath.

More on XPath

In practice, when using XPath, one often needs to prefix the location path with a pointer to the given XML document; e.g.,

```
doc("gazetter.xml")//Feature[@type='Mountain']/text()
```

Other features in XPath include: navigation based on document order, position and size of context, treatment of namespaces, a rich language of expressions.

For full details on XPath and XQuery see the W3C specification:

`http://www.w3.org/TR/xpath`

A tutorial can be found at:

`http://www.w3schools.com/xpath/`