CSE 344 Homework 5: XML

Objectives:

- To be able to manipulate XML: query it with XQuery.

Assignment Tools:

XQuery via Saxon (which has both Java and .NET versions).

1. Please first install saxon on your pc (the Saxon Java version only requires unzipping the jar file).
   - Click Saxon-HE
   - Click 9.4
   - Look for "Looking for the latest version? Download SaxonHE9-4-0-6J.zip (2.8 MB)" or whatever is the latest version when you do his homework.
2. Download the Mondial XML dataset from here.
3. Download the Mondial DTD from here.
4. Follow the brief tutorial to get started (Saxon with Java+Linux). Note: you may have already finished steps one and two.

Extra tool:

- You may want to use xmllint for pretty-printing XML.

References:

- Link to the Saxon command line reference page.
- General Documentation for Saxon.
- XQuery reference.
- XQuery functions reference (Please don't use any functions that Saxon doesn't support).

Due date:

Thursday, February 20, 2013, at 11:59 pm

Drop box

Turn in here

Turn in format:

Turn in each XQuery in a separate file called "Problem[problem #].xq". For example, for problem 1 you turn in a file called "Problem1.xq". In addition, for problem 6 turn in your resulting html file called "Problem6.html". Each XQuery file should start with your name and course in a coment, and should end with the first 3 items returned (if there are fewer than 3, list them all) also in comments. XQuery comments look like `(: this :)`. For example, for the first question (*Retrieve all the names of all cities located in Peru, sorted alphabetically*), your file should be:

```
(: Name
   CSE 344
   Other metadata...
:)

(: Problem 1. :)

(Insert your XQuery here)

(: Results
    <result>
```

```
                <country>
                    <name>Peru</name>

                    <city>
                        <name>Abancay</name>
                    </city>
                    <city>
                        <name>Arequipa</name>

                    </city>
                    <city>
                        <name>Ayacucho</name>
                    </city>
                    ...
                </country>

            </result>
        :)
```

We will run your XQuery files. Please make sure they run, by trying out commands like this:

```
java -cp saxon9he.jar net.sf.saxon.Query ProblemX.xq
```

Problems

[**80 points**: 10 per question (8 for correct answer, 2 for following the DTD), plus 10 pts for the correct HTML file in problem 6]

Consider the XML data instance (the same dowloaded in setup) mondial.xml, available here. Write XQueries to answer the following questions. You need to find out how various elements are nested, e.g. what is under a **country**, under which element is a **city** etc; for that inspect the mondial.dtd (ignore the warning that the data is not valid), or inspect mondial.xml directly. Your XQuery output should follow the associated DTD provided at each question. We will inspect visually if your output follows the DTD, except for problem 7, where we will validate your output automatically. Furthermore, the output of each XQuery should be a well formed XML after standard XML headers (<?xml version="1.0" encoding="UTF-8" ?>, etc) have been added. That is, the output of the first question should be (along the lines of):

```
<result>
    <country>
        <name>Peru</name>

        <city>
            <name>Abancay</name>
        </city>
        <city>
            <name>Arequipa</name>

        </city>
        <city>
            <name>Ayacucho</name>
        </city>
        ...
    </country>

</result>
```

The amount of white space does not matter.

For problem 7 you need to validate your results with the appropriate DTD using the w3 markup validator; instructions for how to use this validator are provided here.

1. Retrieve all the names of all cities located in Peru, sorted alphabetically.

```
<!ELEMENT result (country)>
<!ELEMENT country (name, city+)>
<!ELEMENT city (name)>
<!ELEMENT name (#PCDATA)>
```

2. Find all countries with more than 20 provinces. Order descending by the number of provinces.

```
<!ELEMENT result (country*)>
<!ELEMENT country (name)>
<!ATTLIST country num_provinces CDATA #REQUIRED>
```

```
<!ELEMENT name (#PCDATA)>
```

3. For each province(state) in the United States, compute the ratio of its population to area, and return each province's name, its computed ratio, and order them descending by the ratio.

```
<!ELEMENT result (country)>
<!ELEMENT country (name, state+)>
<!ELEMENT state (name, population_density)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT population_density (#PCDATA)>
```

4. Find all the provinces(states) of the United States with population more than 11,000,000. Compute the ratio of each qualified state's population to the whole population of the country. Return each state's name and the ratio. Order by the ratio in descending order.

```
<!ELEMENT result (country)>
<!ELEMENT country (name, state+)>
<!ELEMENT state (name, population_ratio)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT population_ratio (#PCDATA)>
```

5. Find the names of all countries that have at least 3 mountains over 2000m high, and list the names and heights of all mountains in these countries (regardless of their height). Note: the height attribute is in meters, so you don't have to do any conversions.

```
<!ELEMENT result (country+)>
<!ELEMENT country (name, mountains+)>
<!ELEMENT mountains (name, height)>
<!ELEMENT height (#PCDATA)>
<!ELEMENT name (#PCDATA)>
```

6. For each river which crosses at least 2 countries, return its name, and the names of the countries it crosses. Order desending by the numbers of countries crossed. Place your results into an html file, and verify whether you can/can't view them in your web browser. Turn in the html file along with your query.

```
<!ELEMENT html (head, body)>

<!ELEMENT head (title)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT body (h1, ul)>
<!ELEMENT h1 (#PCDATA)>
<!ELEMENT ul (li+)>
<!ELEMENT li (#PCDATA | div | ol)*>
<!ELEMENT ol (li+)>
<!ELEMENT div (#PCDATA)>
```

The idea with the <li> containing a font and ol tag is such that the output looks roughly like:

```
...
<ul>
  <li>
    <div>River name</div>

    <ol>
      <li>Country crossed #1</li>
      <li>Country crossed #2</li>
      ...
    </ol>

  </li>
  ...
</ul>
```

Note: Use the `country` attribute for the tag `<river>` to find the respective countries.

For this problem you need to turn in two files: `Problem6.xq` and `Problem6.html` (the output of your query).

7. Find the countries adjacent to the 'Pacific Ocean' (sea). For this question you are required to validate your output using the w3 markup validator. We will do this for your answer. Follow the instructions above for validating XML in the W3 Markup Validator to validate your results for this problem. whitespace matters when validating the XML, so do not format your query to include whitespace while validating.

```
<!ELEMENT result (waterbody)>
<!ELEMENT waterbody (name, adjacent_countries+)>
<!ELEMENT adjacent_countries (country+)>
```

```
<!ELEMENT country (name)>
<!ELEMENT name (#PCDATA)>
```

Note: Use the `country` attribute for the tag `<sea>` to find the respective countries.
