CSE-6490B: Assignment #3

Due 6 December 2013

---

1. **General.** *Block that metaphor!* [analogy] (5 points)

   For each of the following, come up with a good answer for the *analogy*. For example, consider

   relational database : SQL :: XML document : _____

   This is to be read as "relational database *is to* SQL *as* XML *is to* _____." A good answer here is "XQuery". Why? A relational database is a collection of data that can be queried with the query language SQL. An XML document is a collection of data that can be queried with the query language XQuery.

   In each case, you may write a brief—one or two sentence—explanation behind your choice, if you feel it is needed.

   a. except : SQL :: _____ : Datalog¬
   b. (2 points) GAV : mediator :: LAV : _____
   c. SQL : structured data :: _____ : semi-structured data
   d. XQuery : XPath :: SQL : _____
   e. SQL : RDBMS :: _____ : Semantic Web

2. **Trees.** *Can't see the trees for the forest.* (5 points)

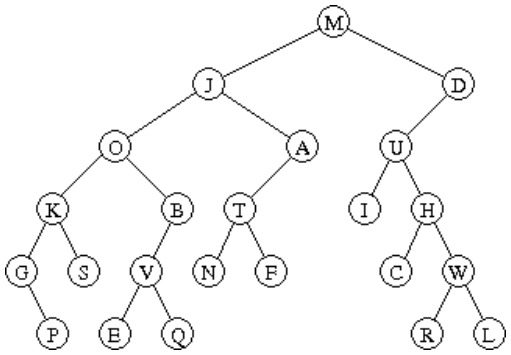
   Dr. Dogfurry, the infamous XML researcher, has the following idea to ship XML "trees" over the network. He suggests flattening the tree to a simple list. Then the tree can be reconstructed on the receiving side. Of course, simply a list of the nodes would not be sufficient to rebuild the tree. It is known that a *binary* tree can be uniquely recontructed given both its pre-order and in-order traversals. So Dr. Dogfurry suggests sending both the pre-order and in-order traversals as lists. (This is just for binary trees. The next step in the research for Dr. Dogfurry is to consider arbitrary trees.)

   On the receiving end, we need an algorithm to reconstruct the binary tree. Write in pseudo-code—in the language style of your choice—an algorithm to do the following. It takes as input two arrays, `Pre` and `In`. The first array represents the pre-order traversal of some binary tree. The second array represents the in-order traversal of the same binary tree. Its output should be a natural data-structure that represents the binary tree.

   For example, consider the following pre-order and in-order arrays:

   pre-order: `M J O K G P S B V E Q A T N F D U I H C W R L`
   in-order: `G P K S O E V Q B J N T F A M I U C H R W L D`

   The only binary tree that results in these pre-order and in-order traversals is



   Note that your algorithm is for binary trees in general, and not binary *search* trees. The example above resulted in a binary tree that is not a binary search tree. This had to be the case since the in-order traversal of the tree is not in alphabetical order of the labels.

   This is easiest to implement this via recursion.

3. **XQuery.** *So this is where the X-path leads.* (5 points)

   Compose your queries with the *Bibliography* XML document in mind. Your queries ought to be logically correct in that they would still do the job as expected if the Bibliography document were to have many new papers added.

   a. Count the number of papers that appear each year in the Bibliography.

   b. (2 points) Return an HTML list of the authors who appear in the Bibliography. Under each author item, present a sub-list of that author's papers by title and year. E.g.,

      ⋮

      - Halevy, Alon
        - Schema mediation in peer data management systems, 2003.

          ⋮

      ⋮

   c. (2 points) In a list, report each author, and for each author, provide a sublist that enumerates his or her co-authors (other authors who have written a paper together with the author). E.g.,

      ⋮

      - Halevy, Alon
        - Ives, Zachary G.
        - Suciu, Dan

          ⋮

      ⋮

      Order the list and the sub-lists.

4. **XML & XPath.** *Extra marks longed for.* [analysis] (5 points)

   Consider an XML document of size N, so with a size on the order of $\mathcal{O}(N)$. That is, printing it in any reasonable way would take $\mathcal{O}(N)$ bytes.

   Assume each element in the document is size $\mathcal{O}(1)$, not counting the content and sub-trees of the element. Assume each element has $\mathcal{O}(1)$ attributes, each of size $\mathcal{O}(1)$.

   An XPath query on the document returns a *list* of the nodes extracted from the XML document that match the query.

   a. (2 points) What is the length of the list resulting from the XPath query '//*' applied to the document?

   b. (2 points) What is the *size* of the results in total from the XPath query '//*' applied to the document? Consider the worst case.

   c. What is the *size* of the results in total from the XPath query '//*/@*' applied to the document? Consider the worst case.