i206 Spring 2013: Assignment 6

Name______________________________

Question 1: Practice with Concepts from Class

(a) (0.5 points) You have a queue of names. Draw what the queue looks like after the following sequence of operations:

enqueue(Alice), enqueue(Bob), enqueue(Carol), enqueue(Dave), dequeue(), dequeue(), enqueue(Erin), dequeue(), enqueue(Fred).

(b) (0.5 points) You instantiated a new binary search tree and inserted elements into the tree in the following order: 12, 3, 1, 30, 4, 5. Draw the state of the tree at the end of the insertions.

(c) (0.5 points) What is the height of the tree in part (b)?

(d) (0.5 points) You instantiated a new heap data structure and inserted elements into the heap in the following order: 12, 3, 1, 30, 4, 5. Draw the state of the heap at the end of the insertions.

(e) (1 point) Assume you have a hash table whose hash function h(key) = key mod 11 where key is an integer. Assuming a hash table of size 11 and a collision system that uses linked lists has chains in the manner described in class, draw a picture of what the hash table looks like after the following series of (key,value) pairs have been entered into the table:

3: “cat”, 5: “dog”, 30: “fish”, 14: “frog”, 10: “cow”, 25: “ape”, 36: “goat”, 100: “horse”

(f) [1 points] Implement the hash table described in (e) using an array for the main part of the table and lists for the chains. Assume only integers are used as keys. You should support the operations of insert and lookup (don’t worry about delete). Insert in the items shown.

Question 2: Practice with Trees: Coding XML

In i202, you learned a lot about XML. XML is a hierarchical, or tree-like structure. In this assignment, we will make use of one of the several XML packages that are included with the latest version of python. We will use this package to read in a simple XML file into a tree structure and then perform some operations on this structure. The intention is for you to learn a bit more about trees and to gain the beginnings of a useful skill, that of being able to automatically process XML. We’ll use a shortened version of the wine.xml file that you saw with the facetmap assignment in i202, called wineabbrev.xml .

The package we will use is one of the more straightforward ones, and does a lot of the work for you. It is called ElementTree, and you load it in as follows:

```
from xml.etree import ElementTree
```

i206 Spring 2013: Assignment 6

In the reference page below, you will need to acquaint yourself with sections 9.11.1 and 9.11.2: http://docs.python.org/release/3.1.3/library/xml.etree.elementtree.html

ElementTree represents XML in a tree structure consisting of objects of type Element. Each Element represents an XML element and so contains the following attributes, assuming you have an instance of an Element called node:

- The value of the XML tag as a string (called tag, access with node.tag)
- The value of the XML text as a string (called text, access with node.text)
- The XML attributes, stored as a dictionary (called attrib, access with node.attrib)
- The trailing text of the XML element, as as string (called tail, accss with node.tail)
- A list of Elements, which is a list of type Element. You access these by calling the node.getchildren() function.

Note that like most tree representations, the data structure is recursive, in that the children are defined in terms of the object type Element. Note also that XML trees are usually not sorted, so there is no sort order for this data structure.

The one tricky part is that at the top level the tree is of type ElementTree, which has unique functions such as parse(), but that it consists of objects of type Element. To get the first Element, you call the getroot() function. Each child is of type Element, but stored in a list of children.

Something else to know is the notion of a "path" when searching the XML tree. If the hierarchy of tags consists of A>B>C, you can't find item C just by looking for "C". You have to specify the entire path using slash notation, as in "A/B/C". I show an example of this in the sample code.

Remember that you can check the type of an object called obj at the python command line with the function type(obj) and you can see the available operations on that object with the commend dir(obj) .

I have written for you a set of functions that show how to make use of ElementTree and Element to read in, traverse, and look for items in the XML tree. In this question I ask you to do some simple extensions of what I've already done. Subquestion (f) is more advanced, and so is optional for bonus points.

(a) [0 points] Experiment with loading in and parsing the XML file by running the functions I've provided and doing some variations of your own to get familiar with this package.

(b) [1 point] Write code that uses the XML structure to print out all the dessert wines.

(c) [1 point] Write code that prints out all and only the leaf nodes, in depth-first traversal order.

(d) [1 point] Create a new object of type Element that is an additional region in one of the countries, putting it in the **last** position among the other regions for that country. In other words, add that object to the appropriate place in the XML tree as the most recently added item for that country. Show that it is working correctly by printing out the relevant country's information.

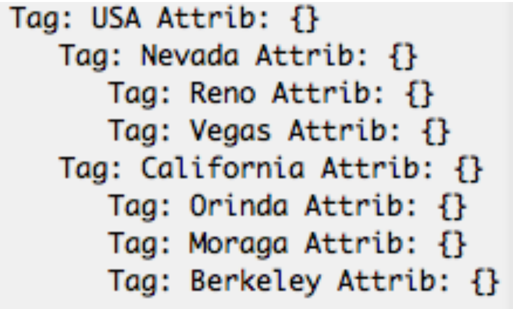
(e) [2 points] Write code to print out the entire tree in breadth-first traversal order.

(f) (**Optional, bonus**) [2 points] To really be useful, you should be able to use this package to create new XML files as well as to process existing ones. For this exercise, begin with a file that has information in a hierarchical format similar to what you'd get from a CSV file or spreadsheet. Assume the file contains information only for the XML tags (not the IDs or other attributes). Here is an example, also provided in the materials enclosed with this assignment in a file called `cities.txt` . You should add more hierarchical items to this file (your choice how many) or use your own file with your own hierarchy:

```
USA|California|Berkeley
USA|California|Moraga
USA|California|Orinda
USA|Nevada|Vegas
USA|Nevada|Reno
```

Note that each line includes the top of the hierarchy, then the second level, then the third (you can go down as many levels as you like). If an internal node repeats, it is shown in the hierarchy. This gives you a path to each leaf and makes it easier to process the input.

Read in these lines of text and use them to build up an XML tree that when printed out in preorder looks like the figure below (although the order of siblings doesn't matter). Note that if you want to have a root node at the top that is not from your file, that is ok and probably makes the code easier to write; you don't need to print it out in the final result.



```
Tag: USA Attrib: {}
   Tag: Nevada Attrib: {}
      Tag: Reno Attrib: {}
      Tag: Vegas Attrib: {}
   Tag: California Attrib: {}
      Tag: Orinda Attrib: {}
      Tag: Moraga Attrib: {}
      Tag: Berkeley Attrib: {}
```

I've put together some code to get you started, called `buildXTree()` in the file provided. Use it just as some inspiration; it is not structured in a way that will work for the problem at hand. You need to write the code to read in the lines from the input file and you need to write a function, probably with a recursive call, to make this work. A hint for one way to do this: treat each input line as a tag path; go through that path adding the highest level node first, then the next level, and so on. If the tag is not in the tree yet, insert it. If it is in the tree, just traverse it till you find the part of the path that hasn't been added yet, and add that, recursively.

Also, note that you can call `tree.write("filename.xml")` to write out your final XML code.