CS2515: Algorithms and Data Structures I

Continuous Assessment 2

This lab is part of the formal continuous assessment for CS2515. It focuses on the implementation and use of Binary Search Trees. It will count for up to 10% of the total marks available for this module.

15/11/2019: movielibrary.py changed to fix a bug

1. **Implementing a Movie Catalogue using Binary Search Trees**

   PyFlix wants to implement a library of movie files that are available for users to stream. The library is dynamic, and keeps changing, with movies being added and removed frequently. Users will want to search the catalogue to see if particular movies are available. This assignment is to write a Python class to represent the catalogue using a recursively-defined Binary Search Tree.

   We will start with a class, `Movie`, representing a movie's title, date of release, and running time. This class is given in movielibrary.py and you don't need to change it. You should inspect it closely, though, to see what methods are offered. Note: for the required part of this assignment, we will not be able to represent two different movies with the same title -- the first one added wil be the one that is stored, until it is explicitly removed.

   An outline of a BinarySearchTree class is given in the `movielibrary.py` file. The class specifies a node in a BST, and since the node contains references to the left and right children, this gives a recursive definition of the (sub)tree which has this node as the root. The initialisation method and the instance variables are given. You do not need to modify this initialisation method. The other methods are given in terms of their signature, with some comments on their use. You must provide working code for every method, without changing the default method signatures (i.e. name and input arguments). When we test your code, we will be issuing calls to each of the specified methods. You can add extra methods (public or private) if it helps you to structure your code, but we will not test them with explicit calls.

   The `movielibrary.py` file also includes two extra helper methods which you can use while testing and debugging your code:

   `_print_structure(self)` will print out to the screen a representation of the structure of the tree rooted at this node, one node per line. Each line contains the element at the node, the height, the elements of the two children (or * if no child), and the parent element (or *).

   `_isthisapropertree(self)` will return True if the tree rooted at this node is a properly implemented tree - that is, all parent and child references match up properly. While you are developing and debugging, you are advised to call this method after every addition or removal of a node, as it will quickly help to identify errors. But do not leave these calls in your final submitted code, as they will slow down the execution significantly on the largest test case.

   i. In order to develop and test your code, you will need to create some trees, so I advise you to start by implementing the `add(self, movie)` method. The file includes a simple class method `_testadd()`, which you invoke by typing `BSTNode._testadd()` on the python command line. Make sure you understand what tree should be created, and then test your code by running this method.

   ii. Then implement the `__str__(self)` method, which should create a string representing the inorder traversal of the tree. This will allow you to print out the contents of the trees as you create them, so that you can check whether or not elements have been successfully added or removed.

   iii. Implement the `search(self, title)` method, which will return the movie object called exactly `title` (or `None` if there is no such movie in the tree). Test your code on the tree returned by the `_testadd()` method.

   iv. Implement the accessor methods -- `findmaxnode(self), height(self), size(self), leaf(self), semileaf(self), full(self), internal(self)` -- and test your code.

   v. Now implement the `remove(self, title)` method. You are advised to implement `remove_node(self)` first, using the pseudocode supplied in `moviedictionary.py`, and then use that in your implementation of `remove(self, title)`. Test your code using the class method `_test()`.

vi. Test your code by creating a tree and then adding or removing elements at the command line. Print the tree statistics and print the ordered sequence of elements.

2. **Using the Binary Search Tree**

   Three testfiles are supplied:

   - smallmovies.txt
   - small_repeated_movies.txt
   - movies.txt

   These files are extracted from the Movies metadata dataset provided by the Data Mining website Kaggle, and the larger file contains over 44000 movies issued since 1900. Note that the department has not inspected the names of these movies, nor inspected the content of these movies. They are taken from public data, and only minimal changes have been made to the dataset, to remove some unprintable characters, and to remove some fields from all data entries. The department is not responsible for the titles.

   Using the supplied method `build_tree(filename)`, create binary search trees of all the movies for each of the testfiles. For each file, state the number of unique movies in your binary search tree, the height of the tree, and the minimum height that would be required if the tree was perfectly balanced. For each file, search for movies with the title "Four Lions", "Wonder Woman", "Touch of Evil" and "Delicatessen". For each movie you find in the tree, state the statistics for that node and the ordered sequence of elements in the subtree for that node.

3. If you have your implementation of the library completed, and you have tested it and made sure it is working, and you want a challenge, then adapt your implementation so that it maintains an AVL tree (i.e. a balanced Binary Search Tree), and compare the performance. There are no extra marks for this.

Submission instructions

To submit, send an email to *k.brown@cs.ucc.ie* with your python file as an attachment. The title of your email should start CS2515 CA2. If you are emailing from a personal account, you must include your ID number in the text of the email.

The submission deadline is listed on the Problem Classes page.

- Do ***not*** be tempted to leave this until later in the month before making progress on the assignment. Evey year, people think there is plenty of time, but then run out of time before they can get it completed. Implementing the recursive Binary Search Tree is tricky if you have not done anything like it before, and getting the `remove_node()` method correct is difficult. You will make mistakes, and it will require significant time to debug it.

- You must submit a single file, which must be named `movielibrary.py`. The file must begin with comments containing answers answers to the questions in part 2. Note that they must be commented out, so that they are not treated as code by the python interpreter.

- Make sure that you compile and test your code before submitting, and that you submit exactly the version that you tested. When you are debugging your code, you may find it helpful to print out lots of messages during the operations. Do not leave these debugging print statements active in the code that you submit, as they will slow down the execution.