Jack Micher

12/10/20

CS/IT 200

Lab 9 Writeup

Lab 9 tasks the programmer to create a transcript manager that allows a user to edit and modify a competent undergraduate student transcript, with the ability to add and remove courses, update the attributes of the courses (name, course code, grade, credits, instructor, and semester taught), undo and redo actions while editing the transcript, and save and load entire transcripts using text files. The storage of elements in the transcript consist of a searchable binary tree map (a Red and Black tree in my case), a self-made Course item class to store attributes of courses, and two stacks for undo and redo. Various amounts of attributes and courses are moved around in tuples and lists.

The transcript stores separate courses using a Red and Black Tree data structure. This allows singular course items to be inserted, found, and removed all in a guaranteed O(logn) time for each because of its usage of map keys and tree balancing. This makes adding, removing, and updating happen very quickly when sorting by course codes. We can use a course's code as a key for the course item as the codes are usually unique given that the courses from one university are sorted out in unique codes. You can also iterate the list in a post-order search order to find attributes of certain elements stored within each node of the tree (in O(n) time). In the transcript manager, the "fieldgpa" command searches through each key-value pair to find nodes keys that are equal to a given course code prefix. I have considered using an AVL tree, but comparing it to a Red and Black and seeing how Red and Black stays balanced more often and takes less time to balance overall, it was obvious that a Red and Black tree was the correct choice.

The Undo/Redo function uses the Stack data structure. Most Undo/Redo functions in applications use Stacks as it is able to interact with the most recent action pushed. In the transcript manager, add, delete, and update will push tuples which contain a string stating the action and a clone of a course item before the action is applied, all into an undo stack. Choosing to undo will add the tuples to the redo stack so they can be reverted. Not only does the functionality of a stack coincide with undo/redo functions, pop() and push() are in O(1) time. It would have been harder to implement undo/redo using other list structures. A deque could have been used, but its extra functionality would not have been needed.

The main program consists of taking a user inputted string and accessing different functions of the manager until the user chooses to exit. There is also a method above the main() which turns grades from F to A into GPA floats.