Project 3: TreeSets

Due: Friday Mar 2, 11:59 pm

1 Assignment Overview

For this project you will be implementing a tree set. Your set will be used to track membership in a group. Items will be sorted based on a comparison function supplied to the tree upon creation.

2 Assignment Deliverables

You must turn in completed versions of the following files:

- TreeSet.py

Be sure to use the specified file name and to submit your files for grading via **D2L Dropbox** before the project deadline.

3 Assignment Specifications

Your task will be to complete the methods listed below:

- `__len__`
- `height`
- `insert`
- `remove`
- `__contains__`
- `first`
- `last`
- `clear`
- `__iter__`

Your implementation of `len`, `height` and `clear` should run in constant time. `iter` can take up to $O(n)$ time total to complete an in-order traversal of the items. The other methods must have $O(\log n)$ *guaranteed* time. Your set must use $O(n)$ space.

The `first` and `last` methods should raise a `KeyError` if the set is empty. No other exceptions should be thrown.

The `insert` and `remove` functions return a Boolean value to indicate whether the operation succeeded or failed. Being a set, duplicate items are not allowed.

The sequence of elements generated by your `TreeSet`'s iterator will be in an order determined by a comparison function, `comp`, supplied to its constructor. `comp(a, b)` compares items `a` and `b`, returning a negative value if `a` $<$ `b` in this sequence, a positive value if `a` $>$ `b`, and 0 if `a` $=$ `b`. You should not use the natural ordering (that is the $<$ operator) of the elements as this will prevent other orders from being used (such as sorting elements in reverse order or using a case-insensitive order for strings).

You should include comments where appropriate. In particular, you should describe the overall method used to balance your tree. You must also add documentation for each of the above methods (as well as for any helper functions created) in the same style as for the pre-defined methods.

4 Assignment Notes

- Points will be deducted if your solution has warnings of any type.
- You have been supplied with stubs for the required methods. You must complete these methods to complete the project. You may add more functions than what was provided, but **you may not modify the signatures of the provided methods**.
- You do not have to worry about error checking for valid input. You may assume that the supplied reader function correctly reads the input file.
- You **do** have to worry about accessing elements from an empty tree.
- You must be able to handle duplicate or non-existent elements.
- Remember that sets do not allow duplicate items.
- Implementations for `is_empty` and `repr` have been provided. Note that these rely on the `len` and `iter` functions, respectively, so you may want to complete these functions early.
- You have been provided with some unit tests that can be used to assess your solution. There are additional test cases that will be kept hidden.
- It is your responsibility to check for potential edge cases. The supplied unit tests do not account for all possible valid inputs
- The `comp` property is a function pointer that determines the ordering of objects. You can invoke it with `comp(a, b)` just like you would for a regular function. It returns -1 if $a < b$, $+1$ if $b < a$ and 0 if $a == b$. You should not use the objects' natural ordering.
- For the `iter` method, you may want to use the `yield` keyword.
- The supplied main file will read in the phonetic alphabet. In the case-insensitive order, there should be 26 items which will spell out the phonetic alphabet in order (there are some additional items that are removed before the end). In the string's natural ordering, they will appear in a different order. There will also be 27 items (both 'sierra' and 'Sierra' will be in the set).
- You may not use any of the classes in the `collections` module or the `set` class.