Carleton University
Department of Systems and Computer Engineering
SYSC 2100 — Algorithms and Data Structures — Winter 2023

Lab 9 - Implementing ADT Set Using a Binary Search Tree

Getting Started

1. Review *Important Considerations When Submitting Files to Brightspace*, which can be found in the course outline.

2. Launch Wing 101 and configure Wing's code reformatting feature. Instructions can be found in Appendix A of document, *SYSC 2100 - Style Guide for Python Code*.

   All code you submit for grading must be formatted. If you decide to disable automatic reformatting, make sure you manually reformat your code (Appendix A.2). At a minimum, we recommend that you reformat your file as you finish each exercise.

3. Download lab9_bstset.py from the *Lab Materials* module in Brightspace.

4. Open lab9_bstset.py in Wing 101. Locate these assignment statements at the top of the file:

   ```
   __author__ = ''
   __student_number__ = ''
   ```

   Replace the empty strings with strings containing your name and student number. (Don't modify the variable names.)

5. Your solutions to the exercises must conform to the coding conventions described in *SYSC 2100 - Style Guide for Python Code*.

6. **Important**: if you decide to write a script in lab9_bstset.py, it **must** be placed in an `if __name__ == '__main__':` block, like this:

   ```
   class BSTSet:
       # methods not shown

   if __name__ == '__main__':
       empty_set = BSTSet()
       s = BSTSet([1, 2, 3, 2, 1])
           ...
   ```

   Doing this ensures that the script will only be executed when you run lab9_bstset.py as the "main" module, and won't be executed when the file is imported into another module; for example, the testing/grading program provided to the TAs.

Overview of Class `BSTSet`

A *set* is a mutable, unordered collection of distinct objects.

Class `BSTSet` in lab9_bstset.py contains an incomplete implementation of ADT Set that uses a binary search tree (BST) as its data structure. This class contains definitions of methods `__init__`, `__str__`, `__repr__`, `__len__`, `__iter__` and `size`. The class also has "stub" implementations of methods `__contains__`, `__eq__`, `__or__` and `add`. If you call any of these methods on a `BSTSet` object, Python will raise a `NotImplementedError` exception. You'll complete the implementation of these methods during this lab.

While working on the exercises, we recommend that you write a PyUnit test program (file lab9_test_BSTSet.py). Define a separate test class (a subclass of `unittest.TestCase`) for each exercise.

Do not modify class `_Node` or methods `__str__`, `__repr__` and `__iter__`.

Exercise 1: Method side has complexity $O(n)$ because it traverses the set's BST, counting nodes. Method `__len__` calls `size`, so it also has complexity $O(n)$. Modify `BSTSet` so that `__len__` has complexity $O(1)$.

Use the Python shell to test `__len__` with an empty set. You won't be able to finish testing until you've defined `add`, in Exercise 2.

After you've completed this exercise, you can delete `size` and its recursive helper function, `_size`.

Exercise 2: Read the docstring for `add`, delete the `raise` statement, then implement the method and recursive helper function `_add`. Hint: this method and function are very similar to method `insert` and recursive function `_insert` in file `binarysearchtree.py`. (The algorithms were presented in a recent lecture.) Note that `insert/_insert` allow duplicate values to be inserted in a binary search tree; however, all elements in a set must be distinct (no duplicate elements).

Your solution must be recursive: **don't** translate the iterative `insert` algorithm presented in the lecture slides into Python.

Use the Python shell to run a few tests on `add`. After `add` passes all its tests, test `__len__` with sets that contain more than one element.

Exercise 3: Read the docstring for `__contains__`, delete the `raise` statement, then implement the method and recursive helper function `_contains`. Your solution must be recursive: **don't** translate the iterative `search` algorithm presented in the lecture slides into Python.

Use the Python shell to run a few tests on `__contains__`.

Exercise 4: In order to be able to use Python's == operator to determine if two sets are equal, we need to define a method named `__eq__` in `BSTSet`. Read the docstring for `__eq__`, delete the `raise` statement, then implement the method.

Note that if parameter `other` refers to an `object` that is not a `BSTSet`, the method should return `False`.

Use the Python shell to run a few tests on `__eq__`.

Exercise 5: The *union* of two sets *A* and *B* is a new set that combines the members of *A* and *B*. Python's built-in `set` type uses the `|` operator to represent the union operation. In order to be able to use this operator to form the union of two `BSTSet` objects, we we need to define a method named `__or__` in `BSTSet`. Read the docstring for `__or__`, delete the `raise` statement, then implement the method.

Note that if parameter `other` refers to an object that is not a `BSTSet`, the method should raise a `TypeError` exception that displays the message: "`unsupported operand type for |`".

Use the Python shell to run a few tests on `__or__`.

Wrap-up

- Before submitting lab9_bstset.py, review your code. Does it conform to the coding conventions, as specified in the *Getting Started* section? Has it been formatted? Did you edit the `__author__` and `__student_number__` variables?
- Submit lab9_bstset.py to Brightspace. Make sure you submit the file that contains your solutions, not the unmodified file you downloaded from Brightspace! You are permitted to make changes to your solutions and resubmit the files as many times as you want, up to the submission due date. Only the most recent submission will be saved by Brightspace.
- Solutions that are emailed to your instructor or a TA will not be graded, even if they are emailed before the submission due date.