Programming Abstractions – Homework IV

Dr. Ritwik Banerjee
Computer Science, Stony Brook University

This homework document consists of 3 pages, and involves concurrency (*i.e.*, interleaved execution of multiple tasks) and a procedural abstraction: iteration. You will be using both Java and Python for this assignment.

Development Environment: It is highly recommended that you use IntelliJ IDEA for your Java code, and PyCharm for your Python code. You can, if you really want, use different IDEs, but if things go wrong there, you may be on your own. Plus, IDEA and PyCharm have very similar user interfaces, so using one of them makes it very intuitive to use the other.

Programming Language: Just like the previous assignment, all Java code *must* be JDK 1.8 compliant.[1] Your Python installation must be version 3.6 or above. Make sure this is reflected in your PyCharm project settings as well. In particular, never ever ever ever use Python 2.x (this is "legacy", and will never be used in any current or future software).

1. Design abstraction of iteration

As a programmer, you have probably used an expression like `for (E e : a_list_of_elements)`, especially when working with various collections of objects such as a list of numbers or a set of strings. And of course, we have all used the for-loop on such collections. But how exactly do these expressions work? Why is it that we could not use such readily available loop structures on collections we built from scratch (for example, a tree) in a data structure course?

The answer lies in a high level of abstraction that embodies the basic idea of iteration. In Java, expressions like `for (E e : a_list_of_elements)` or `a_list_of_elements.forEach()` are allowed for all iterable objects. But how does an object convey that it allows iteration? It does so by implementing the `Iterable` type.

1. Your first task is to read the Java documentation of the `Iterable` interface and understand how it works (you can ignore the "spliterator"): docs.oracle.com/javase/8/docs/api/java/lang/Iterable.html (30)

   Use this knowledge to build a binary search tree. Your binary search tree must be parameterized, and it must implement the `Iterable` interface (again, parameterized).

   The iteration order on your tree should be such that the order of iteration traverses its elements in increasing order: starting with the least value, and ending with the highest value. Like the previous assignments, the internal details of the implementation are left to you. But the following "user code" should work and provide the output as shown next.

```
public static void main(String... args) {
  // each tree has a name, provided to its constructor
  BinarySearchTree<Integer> t1 = new BinarySearchTree<>("Oak");
  // adds the elements to t1 in the order 5, 3, 0, and then 9
  t1.addAll(Arrays.asList(5, 3, 0, 9));
  BinarySearchTree<Integer> t2 = new BinarySearchTree<>("Maple");
  // adds the elements to t2 in the order 9, 5, and then 10
  t2.addAll(Arrays.asList(9, 5, 10));
  System.out.println(t1); // see the expected output for exact format
  t1.forEach(System.out::println); // iteration in increasing order
```

[1] You may have a higher version of Java installed, but the "language level" must be set to Java 8. This can be easily done in IntelliJ IDEA by going to "Project Structure" and selecting the appropriate "Project language level". This is a very important requirement, since Java 9 (and beyond) has additional language features that will not compile with a Java 8 compiler.

```
    System.out.println(t2); // see the expected output for exact format
    t2.forEach(System.out::println); // iteration in increasing order
    BinarySearchTree<String> t3 = new BinarySearchTree<>("Cornucopia");
    t3.addAll(Arrays.asList("coconut", "apple", "banana", "plum", "durian",
                            "no durians on this tree!", "tamarind"));
    System.out.println(t3); // see the expected output for exact format
    t3.forEach(System.out::println); // iteration in increasing order
}
```

Expected output:

```
[Oak] 5 L:(3 L:(0)) R:(9)
0
3
5
9
[Maple] 9 L:(5) R:(10)
5
9
10
[Cornucopia] coconut L:(apple R:(banana)) R:(plum L:(durian R:(no durians on this tree!)) R:(tamarind))
apple
banana
coconut
durian
no durians on this tree!
plum
tamarind
```

2. Merging two binary search trees

For this section, recall the "merge" algorithm from merge sort. There, we already have two sorted sublists, which are merged to form a sorted list. Also recall that this merge algorithm had linear time complexity, requiring only one pass over the two sublists. Here, instead of sublists, we have trees (whose iteration order ensures that elements are traversed in a min-to-max sorted order). Thus, we can apply the same algorithm to merge two trees to build a sorted list of their elements.

2. Write a method in the same `BinarySearchTree` class, with the following signature (40)

```
public static <T extends Comparable<T>> List<T> merge(List<BinarySearchTree<T>> bstlist)
```

This method must be multi-threaded: if there are k trees in `bstlist`, the iteration work over these trees must be evenly distributed across k threads (created in addition to the already existing main thread). For example, if there are 4 trees to be merged, there should be 4 threads created, each responsible for iterating over a binary search tree. Note that they *all write to the same list, which is eventually returned by the method*.

This is essentially a generalization of the merge algorithm you studied in merge sort. Instead of merging two lists or arrays, you are now merging k of them. In the original version, you (1) maintained references to the least elements of two sublists (or "half arrays" to be merged), (2) compared them, (3) put the lower element into the output array, and (4) updated the reference before going back to step 2. Here, instead of maintaining two references, you will have k references, while the rest of the merging logic remains the same.

For simplicity, you can assume that this code is not expected to handle more than 5-6 threads (in addition to the main thread).

3. Iterations in Python

By this point, the conceptual aspects of iteration are hopefully quite clear. So, we will apply those same concepts and implement a solution to the same problem in Python. Implementing the notion of making a class *comparable* requires additional modules to be imported for decoration, so we will only use integer values in our binary search tree nodes in Python.

3. Write your code in a Python file called "`binarysearchtree.py`". The following is expected to print `[Oak] 5 L:(3 L:(0)) R:(9)` (same format as in the previous Java example). (30)

```
if __name__ == "__main__":
    tree = BinarySearchTree(name="Oak", root=Node())
    tree.add_all(5, 3, 9, 0)  # adds the elements in the order 5, 3, 9, and then 0
    print(tree)
```

To make your tree iterable, you must implement the method

- `__iter__()`, which returns the iterator object and is implicitly called at the start of loops.

This method should be a generator of the node values (and thus, a coroutine). On one hand, this allows for iteration (just like you saw in the Java portion of this assignment). On the other hand, after yielding each value, it pauses its own flow (and resumes again, *from that point where it had paused*, when called the next time).

Just like the Java portion of this assignment, the following "user code", when added to your module, should run and iterate over the trees to print the elements in increasing order.

```
if __name__ == "__main__":
    t1 = BinarySearchTree(name="Oak", root=Node())
    t2 = BinarySearchTree(name="Birch", root=Node())
    t1.add_all(5, 3, 9, 0)
    t2.add_all(1, 0, 10, 2, 7)
    for x in t1.root:
        print(x)
    for x in t2.root:
        print(x)
```

- Please keep in mind these homework-related points mentioned in the syllabus.
- **What to submit?** The complete codebase, which consists of: (i) `binarysearchtree.py`, and (ii) `BinarySearchTree.java`.
- You can import libraries in your code as long as they satisfy ALL these conditions: (i) they are a part of core Java/Python, (ii) they do not provide an implementation of the kind of iteration you are required to implement, and (iii) they do not provide an implementation of any data structure you are required to implement. If you are unsure about a specific method from a library being allowed, please immediately ask on Piazza.
  Deviations from the expected submission format and the above requirements carries varying degrees of score penalty (depending on the amount of deviation).

Submission Deadline: May 9 (Monday), 11:59 pm