Testing Report (User 36)

Overview

Methods that Passed:

- insert()

Methods that Failed:

- toString()
- remove()
- split()
- join()
- iterator()
- Custom edge cases

Testing Methodology

insert()

I tested this method by first inserting random integer keys and values into your TreapMap. I stored the values I inserted in a hash map. After inserting 1000 elements, I made sure that every element was found in the HashMap was found in the TreapMap and vice versa. I also made sure that the treap satisfied the binary tree property and the heap property.

remove()

I tested this method similarly to how I tested *insert()*, except I gave the treap a 33% chance of removing an element. I made sure that when an element was removed, when trying to look it up again, null would be returned. I did the same checks as I did in *insert()*.

split() and join()

I tested this method by inserting elements randomly into the TreapMap with sizes of powers of 2 until 1024. I split the Treap, made sure that the left Treap had elements strictly less than the key, and made sure that the right Treap had elements greater than or equal to the key. I also made sure both Treaps had the same keys as the original Treap. I then joined the subtreaps and made sure it had the same elements as the original treap. For the subtreaps and the joined treap, I made sure they satisfied the binary tree property and the heap property.

iterator()

I tested this method by constructing a Treap similarly to the Treap constructed in remove() and made sure that each element as it was iterating was greater than its previous element. I made sure that each element that was iterated through was present in the TreapMap. I also made sure that ConcurrentModificationException was thrown when the TreapMap was altered. I made sure that operations that did not affect the TreapMap, such as removing a node that wasn't found or passing in nulls for all operations, did not affect the iterator.

toString()

This was indirectly tested through my testing of the heap property and the bst property. I built a parser to build a tree from string output.

Specific Edge Cases

The edge cases I tested were as follows. I made sure that on an empty Treap, insert and join returned null while split led to an array of empty strings when calling *Arrays.toString()* and join with an empty Treap led to another empty Treap. I also made sure that when you pass in null to any of these methods, the methods return null and does not alter the Treap. I also made sure that when all entries were removed, the treap was properly reset to an empty Treap.

Notes

- Your program failed the toString() test due to having an extra "\r\n" character at the end of the output. Also, for a design decision, you could use System.lineSeparator() instead of "\r\n" to use the appropriate next line character for the OS that is running your code (design decision so you don't need to do this).
- For the remove method of your treap, your treap could not find an element that should have been present in your treap. This is possibly due to incorrectly deleting a node, such as when it only has one child, since you're insert() method seems to be working properly. In addition, sometimes your treap would not satisfy the heap property when deleting a node.
- When splitting a treap, the subtreaps would not contain all the keys of the original treap. When the subtreaps were joined, the joined treap would not satisfy the heap property.
- For your iterator, the only real problem was failure to throw a ConcurrentModificationException when the Treap was modified.
- When trying to remove a null key when the treap only has one element in it, your treap deleted the element in the treap.
- A node with a null key was inserted into the treap.
- A node with a null value was inserted into the treap.
- Recommendations:
    - Since there are issues with finding nodes after deletions, make sure that your pointer manipulation is correct. Since your insert method seems to be fine, it may be due to your deleting a leaf algorithm.

Below is graphs for further analysis of runtime and other miscellaneous features tested empirically: