LAB 2: RED BLACK TREE & TREEMAP

INTRODUCTION

RED BLACK TREE

A redblack tree is a kind of self-balancing binary search tree in computer science. Each node of the binary tree has an extra bit, and that bit is often interpreted as the color (red or black) of the node. These color bits are used to ensure the tree remains approximately balanced during insertions and deletions. Balance is preserved by painting each node of the tree with one of two colors in a way that satisfies certain properties, which collectively constrain how unbalanced the tree can become in the worst case. When the tree is modified, the new tree is subsequently rearranged and repainted to restore the coloring properties. The properties are designed in such a way that this rearranging and recoloring can be performed efficiently.

TREEMAP

A Red-Black tree based NavigableMap implementation. The map is sorted according to the natural ordering of its keys, or by a Comparator provided at map creation time, depending on which constructor is used. This implementation provides guaranteed log(n) time cost for the containsKey, get, put and remove operations. Algorithms are adaptations of those in Cormen, Leiserson, and Rivest's Introduction to Algorithms.

REQUIREMENTS

RED BLACK TREE

1. 1.getRoot: return the root of the given Red black tree.
2. isEmpty: return whether the given tree isEmpty or not.
3. clear: Clear all keys in the given tree.
4. search: return the value associated with the given key or null if no value is found.
5. contains: return true if the tree contains the given key and false otherwise.
6. insert: Insert the given key in the tree while maintaining the red black tree properties. If the key is already present in the tree, update its value.
7. delete: Delete the node associated with the given key. Return true in case of success and false otherwise.

TREEMAP

1. ceilingEntry: Returns a key-value mapping associated with the least key greater than or equal to the given key, or null if there is no such key.
2. ceilingKey: Returns the least key greater than or equal to the given key, or null if there is no such key.
3. clear: Removes all of the mappings from this map.
4. containsKey: Returns true if this map contains a mapping for the specified key.
5. containsValue: Returns true if this map maps one or more keys to the specified value.
6. entrySet: Returns a Set view of the mappings contained in this map in ascending key order.
7. firstEntry: Returns a key-value mapping associated with the least key in this map, or null if the map is empty.
8. firstKey: Returns the first (lowest) key currently in this map, or null if the map is empty.

9. floorEntry: Returns a key-value mapping associated with the greatest key less than or equal to the given key, or null if there is no such key.
10. floorKey: Returns the greatest key less than or equal to the given key, or null if there is no such key.
11. get: Returns the value to which the specified key is mapped, or null if this map contains no mapping for the key.
12. headMap: Returns a view of the portion of this map whose keys are strictly less than toKey in ascending order.
13. headMap: Returns a view of the portion of this map whose keys are less than (or equal to, if inclusive is true) toKey in ascending order..
14. keySet: Returns a Set view of the keys contained in this map.
15. lastEntry: Returns a key-value mapping associated with the greatest key in this map, or null if the map is empty.
16. lastKey: Returns the last (highest) key currently in this map.
17. pollFirstElement: Removes and returns a key-value mapping associated with the least key in this map, or null if the map is empty.
18. pollLastEntry: Removes and returns a key-value mapping associated with the greatest key in this map, or null if the map is empty.
19. put: Associates the specified value with the specified key in this map.
20. putAll: Copies all of the mappings from the specified map to this map.
21. remove: Removes the mapping for this key from this TreeMap if present.
22. size: Returns the number of key-value mappings in this map.
23. values: Returns a Collection view of the values contained in this map.

TIME ANALYSIS

RED BLACK TREE

1. getRoot: $O(1)$, the root is always known for the tree
2. isEmpty: $O(1)$, if there is no root then there is no tree which uses
3. clear: $O(n)$ because you don't need to look for the node you'll delete
4. search: $O(\log n)$ in worst and average case scenario.
5. contains: it uses "search" first then it also takes up $O(\log n)$.
6. insert: $O(\log n)$ in worst and average case scenario.
7. delete: the node has to be searched first then after that deleted therefore it is $O(\log n)$ too.

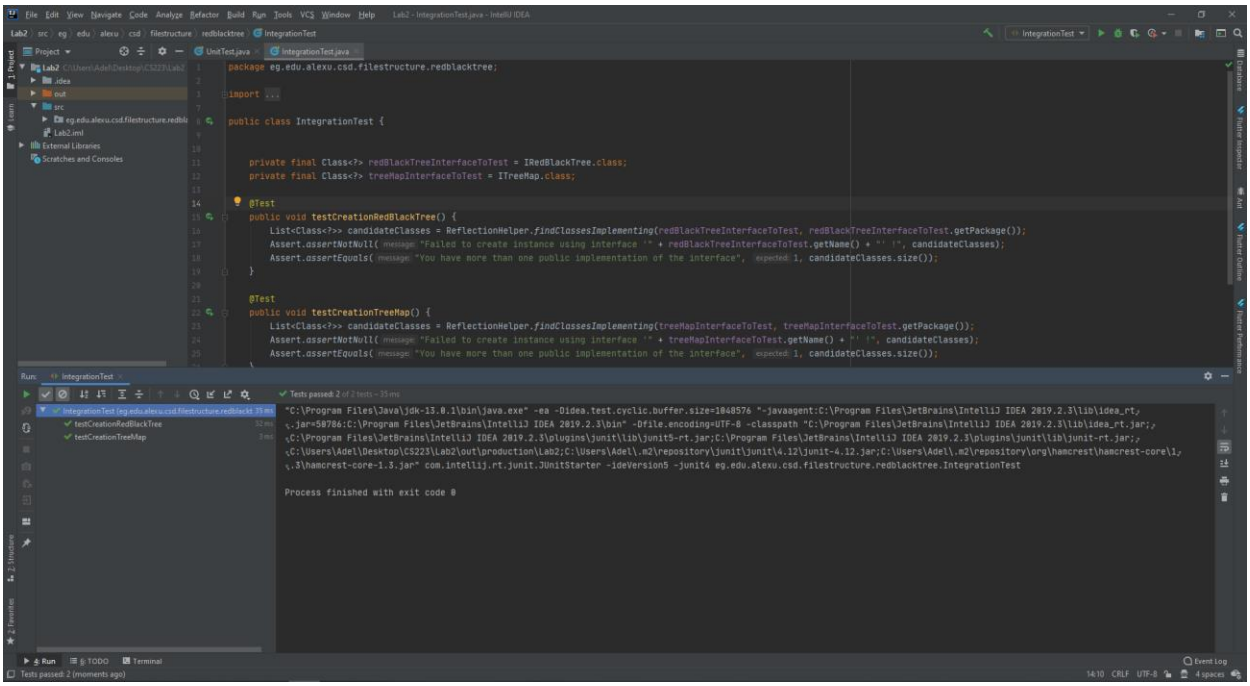
TREEMAP

1. ceilingEntry: $O(\log n)$ to search for the ceiling entry (found using ceiling key).
2. ceilingKey: $O(\log n)$ to search for the for the equal key or if not found then the least greater than key.
3. clear: $O(n)$ because you don't need to look for the node you'll delete
4. containsKey: $O(\log n)$ just to for searching a node
5. containsValue: $O(n \log n)$ in the average case scenario.
6. entrySet: $O(n)$ to get all entries
7. firstEntry: $O(\log n)$ to find the first entry.
8. firstKey: $O(\log n)$ to find the first key.
9. floorEntry: $O(\log n)$ to search for the ceiling entry (found using ceiling key).
10. floorKey: $O(\log n)$ to search for the for the equal key or if not found then the greatest less than key.

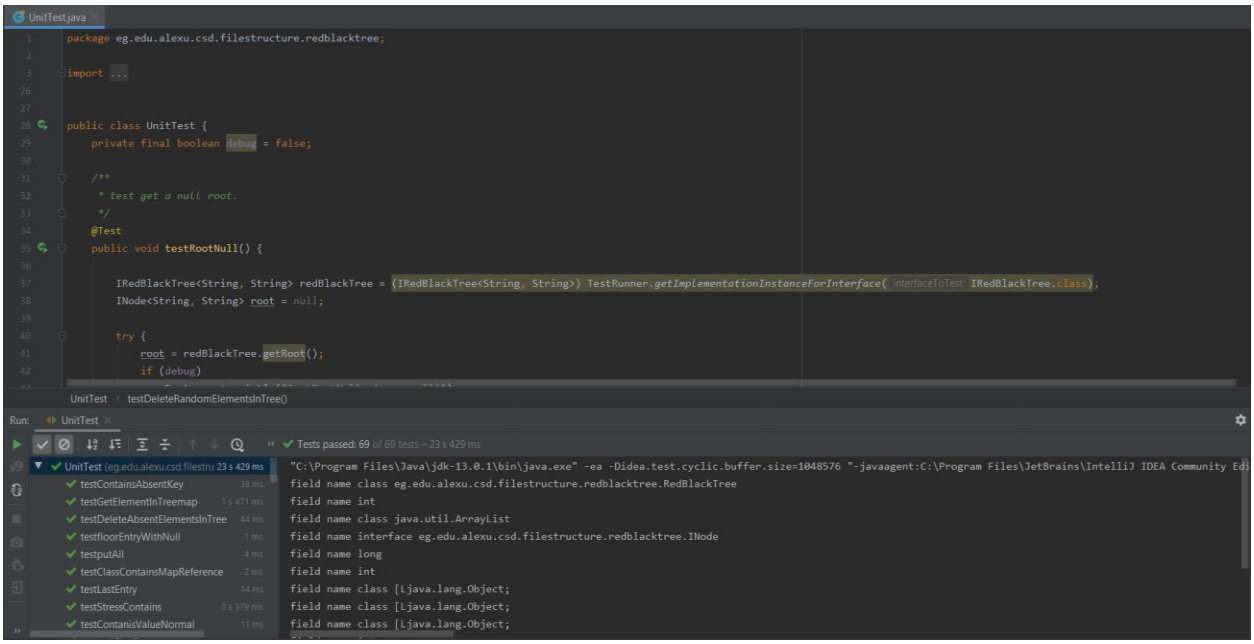
11. get: $O(\log n)$ to search for the node
12. headMap: $O(\log n)$ to find the node and each node that is less than it you put it in the map
13. headMap: $O(\log n)$ to find the node and each node that is less than it you put it in the map + the node.
14. keySet: $O(n)$ to get all keys
15. lastEntry: $O(\log n)$ to find the last entry.
16. lastKey: $O(\log n)$ to find the last key.
17. pollFirstElement: $O(\log n)$ to search and then delete
18. pollLastEntry: $O(\log n)$ to search and then delete
19. put: $O(\log n)$ as the normal insert
20. putAll: $O(n)$ for all node
21. remove: $O(\log n)$ to search for the node and then delete it
22. size: O(1) there is a pointer about the size
23. values: $O(\mathrm{n}\log n)$ to get all the values from the keySet function

TEST SCREENSHOTS:

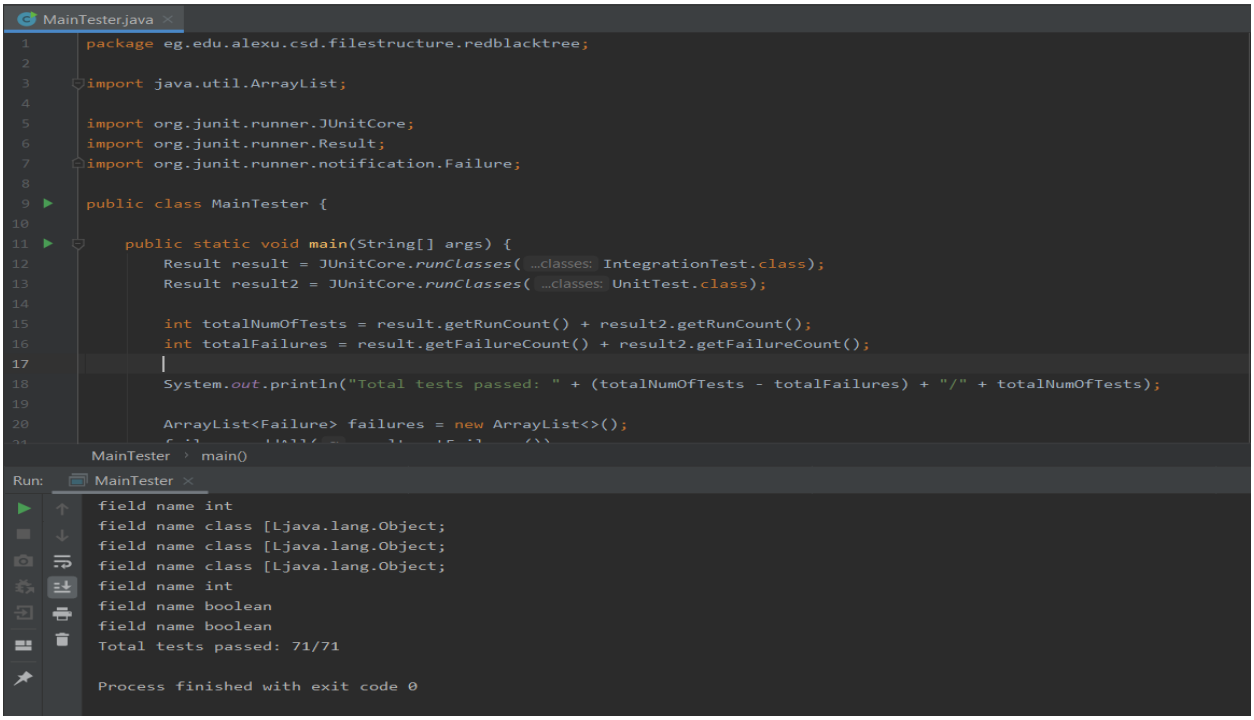
INTEGRATIONTEST



UNITTEST



MAINTESTER



STUDENTS:

- Seif El-Deen Ehab Mostafa Ibrahim - #33
- AbdelRahman Adel AbdelFattah AbdelRaouf Mahmoud - #37