CSCI 3901 Assignment 2(Problem 2)

Name : Yogish Honnadevipura Gopalakrishna(B00928029)

Overview

This program executes AmortizedTree which has two data structures in it. The first one is the tree which has nodes and the second one is an array namely “valuesToBeAdded”.

The class “AmortizedTree” has implemented two interfaces namely “Searchable” and “TreeDebug” which has the following methods:

Searchable

Add : This method takes a String namely “key” as its parameter and checks whether the key is null, empty or non-unique in the AmortizedTree. If any of these checks is true, the method returns false. Else, it will do the following steps:

- Convert the key into lowercase because string comparisons should not be case sensitive
- Check if the array is full
    - Sort the array and then insert it into the tree so that it is as balanced as possible
    - Reinitialize the array based on ceil(log(size_of_tree)) value
    - Put the key into the 0th index of the newly initialised empty array
    - Return true
- If the array has space in it,
    - Retrieve the index of the space in the array
    - Add the key into that index
    - Return true

Find : Searches for the key passed as parameter in the tree and the array, if found, it returns true else the method returns false.

Remove : Below method removes the key from the AmortizedTree if the key is present and returns true, else returns false. It uses the following logic for this implementation,

- Search in the tree, if found there are 3 cases to consider
    - If the found node has no child, remove the node
    - If the found node has either left or right child, make them as successors
    - If the found node has both children, the node which is present in the leftmost position of the right child of the found node will be its successor.
- If it's in the array, then traverses through the array and removes the element by replacing the key by the elements to its right hand .This shifting process starts from the index where key is found and done until the end of the array.

Size : This method finds the size of the tree. Then, it finds the number of elements in the array where the null values are excluded. It then returns the size of tree + size of array.

Rebalance : This method rebalances the unbalanced tree. This is done by taking all the values of the string into an array, sorting them using the idea "inserting the middle element first and then middle of what is left to either side". After the array is sorted, insert them into the tree.

RebalanceValue : This method also gets the values of the tree inside an array and sorts them. Then, the root node will be given the value of the "key" which is passed as the parameter. Now insert the elements of the array into the tree, and the key is not inserted again as there is a duplicate check in the method.

TreeDebug

printTree: This method uses in-order traversal to print the elements of the tree in the ascending order. It uses StringBuilder to store the value and the depth of each element in the tree.

awaitingInsertion: This method traverses through the array and returns the number of elements in the array where the null values are excluded. Then a new array is initialised using the retrieved size. Now it copies the value of the array into the new array until the index where null is present, and then the new array is returned.

treeValues: This method also uses in-order traversal to append the elements of the tree in the ascending order into the StringBuilder where each element is separated by a space. Now by using regex we copy the values into an array and return the same array.

depth: Searches for the argument “key” in the tree, if found returns its level. If not found, search for the key inside the array by traversing through the array, if found, it returns its index * -1, else it returns size of the tree * 2.

Files and external data

There are six main files:

-Searchable.java -> An Interface which contains abstract methods to perform tree operations.

-TreeDebug.java -> An Interface contains abstract methods to search or return tree elements in different ways.

-AmortizedTree.java - >contains implementation methods of both interfaces.

-Helper.java -> contains methods to perform sort operations on arrays and calculate log Value.

-TreeOperations.java - contains methods to perform Tree operations. These methods are Used in AmortizedTree.

- TreeNode.java - contains a single node data in it.

Data structures and their relations to each other

- This program has an array and a binary search tree in the Amortized tree, which holds the values before they are inserted into the tree.
- The array is designed to reinitialise automatically based on the number of elements in the array
- When the array is full and a new element is to be put into the array, it is when the elements in the array are inserted into the tree and then the array will be given a new size and the new element will be inserted at 0th index.
- The array is sorted based on the “middle elements first” idea before elements or values are inserted into the tree. Therefore, the tree can be as balanced as possible.
- The binary search tree or the array cannot have duplicate elements in them combined.
- Operations like add, remove, search, print are performed on the tree and the array

Choices

- When an array is full and overflowed, all the elements inside the array are inserted into the tree before the array is reinitialised and given a new size and the new element will be positioned at arr[0] of the new array.
- When there are even numbers in an array, there will be 2 middle elements. I have taken the bigger element of these two as the middle element.

Key algorithms and design elements

The algorithm and design used for implementing the Amortized Tree is as follows:

1. Add an element which goes into the waiting array
2. Add another element which overflow the array, inserts the first element into the tree and second element into the array
3. When the tree size is 3, the array size is incremented to two.
4. When the array is full and the third element is added, it then overflows the array and adds the existing element of the array into the tree and now, the tree size is 5 which increases the array size to 3.
5. When an element is deleted the array is not reinitialised but waits for the array to overflow to reinitialise the array.
6. The depth of the root is taken as 1 and the depth increases as the tree is traversed.

Limitations

- When the root is null and initial values are added into the array, it may lead to tree imbalance until the rebalance method is called.
- When an array is full and overflowing, all the elements of the array are inserted before reinitialising the array based on the number of elements in the tree.

References

https://stackoverflow.com/
https://www.geeksforgeeks.org/
https://www.w3schools.com/