Report

Data Structure

For this project, we chose a binary search tree structure to manage data. Because, we needed to complete many search operations to finish methods. In this case, Binary search tree was a good choice for us instead of array, hash map or linked list data structure.
Because, linked list and array are so straightforward and its performance for searching is not well in large scales. The performance of array structure could be improved with some search algorithms but it is not offering too much for this project, at least for us. Although hash map has better performance for search, it takes too much space and hard to manage data with a proper way. So, bst(binary search tree) was a better choice to complete this project with a proper algorithm with our current programing knowledge.

Bst takes between O(log2n)-O(n), where n is number of elements, according to order of input. In a balanced tree worst case is O(log2n) but worst-case scenario for binary search tree is same with linked list, O(n). So, making a self-balancing binary search tree(avl tree) algorithm would be best for us.

An Overview to Methods

add(AnyType item)

We create bst structure in this method by inserting items with this rule: Greater values stores in right side and small values stores in left side. To apply this rule, program compares values of node and the given data, according to comparison goes left or right in the tree. First of all, program visits node by applying the rule and checks if there is an element in tree with the same value of given data. İf there is, program increases item counter of related node. If there is not, program creates a new node and connects it to tree according to rule and increases distictSize of bag. Also, program increases sizeOfBag for each run, by this way we get a variable which stores number of elements in the bag.

findElement(AnyType data)

findElement method starts from root and compares value of root with given item. If item is bigger than root value than method changes its current node to its right, if value is less than root value, method changes its current node to its left child. Because, in a bst big values stores in right child and small values stores in left child. If values match in current node with given data than method returns this node. If it does not match than find element returns null.

contains(AnyType item)

Program searches the tree by using findElement method. After that, program checks if returned value is null or not. If it is null, it returns false because that means this item is not present in tree. If it is different from null, a node , than method returns true.

distictSize()

In our program, it directly returns the distictSize. We required to give the number of different elements in collection. In order to do that, we create a field in bag class named distictSize whose type integer.In our program, every node has a field item_counter which stores number of the element in the node. So, if you add a data which already in collection, program just increases the item counter of the node which stores this data but does not create a new node. So, if we increase distictSize by one when we create a new class, we get the number of different elements in the bag.

elementSize(AnyType item)

Method finds the node by using findElement method and returns its item counter which stores number of this element. If there is no such a node in tree, it returns 0.

isEmpty()

Method checks whether root is null or not. If root is null, that means tree is empty. In this case returns true. If root is not null than returns false because that means there is at least one element in tree so tree is not empty.

size()

Method returns sizeOfBag. Program increases sizeOfBag each time we call add(AnyType item) method. So, it is equal to number of elements in the bag.

toString()

Checks tree is empty or not. If tree is empty, it returns a string that says there is no element in the bag. If tree is not empty, program calls a method whose sign is getString(Node<AnyType> node) . This method creates a string and visits every node in inorder sequence. When traveling nodes, method adds every value of node to string as many times as its item counter value. Ex. A node whose data is “blue” and its item_counter is 2. This method will add this node’s data to string two times {“blue”,”blue”,}. After that, getString method returns the string and also toString returns this string.

remove(AnyType item)

Finds the element in tree with the help of findElement method.Checks its item counter if it is greater than 1 than decrease its item counter. If its equal to 1, calls deleteNode method for this node.
DeleteNode method deletes this node from tree without changing tree's structure. Program decreases distictSize if a node is deleted. Program decrease sizeOfBag for every successful attempt. If findElement returns null, remove method returns false and doesn't change anything because there is no such an element in tree.

clear()

Deletes all nodes in tree by calling clearTree method. Changes sizeOfBag and distictSize values to 0 because deleteNode method directly deletes given node and ignores item numbers. clearTree method calls deleteNode method for each node in preorder sequence. By this way, program doesn't miss any data.

equals(Object obj)

1. Checks whether given object and this object is same or not. (In case, item is comparing with itself.)

2. If not checks is this object is a bag or not. If it is not, returns false. If object is not a bag than there is no way it can be equal to this bag.

3. If object is a bag than program checks class types are same or not. If object doesn't contain same type of values with bag than there is no way it can be equal to this bag. If class types are different returns false.

4. if class types are same,checks sizeOfBag and distictSizes are equal or not. If size of bags and distictSize's are different than there is no point to check items, these two definitely different.

5. If two same type bag have same number of elements and distictSize than we should compare objects item by item. To do that, program generates a string for each bag and checks its equality by using equals method of string class.

Ekin Şuataman