COL106 Assignment3
Name: Dipen Kumar
Entry No: 2018CS50098
Group Number: 3

Imports used-

1. import java.lang.Math
2. import java.io.IOException
3. import java.io.FileReader;
4. import java.io.BufferedReader

Interfaces implemented-

1. public interface MyHashTable_<K,T>
2. public interface Student_

Note: I have implemented MyHashTable_<K,T> twice as instructed.

1. MyHashTableDH<K extends comparable<K>,T> for Double Hashing
2. MyHashTableSCBST<K extends comparable<K>,T> for Separate Chaining

MyHashTableDH<K extends comparable<K>,T>

This class will create a array of class NodeDH<K,T> of size as received from constructor.

[**NodeDH<K,T>** is a class having two fields of type K,T i.e. generic field. Its constructor initializes these two field. There also exist two methods that returns the value of these two fields respectively.]

I also have two method of return type Long for h1 and h2 as given to us. Addition to this I also have six more methods as specified in MyHashTable_<K,T>.

1. public int insert (K key,T obj)
   Time complexity- O(n)
   Approach (proof)- I first check that is this key exist already or not. If it exists then return -1 as I am not allowing any duplicate entry. Else I hash the key using (h1+ih2) method and the value I get is the index number of array where I am supposed to store K key and T obj in form of NodeDH<K,t>. If the index is already occupied than iterate further. If we have iterated over entire array and not found any index which is null then return -1 i.e. error. Else return i. Hence in worst case I have gone through entire array list to check whether it is contained in array or not also I have gone over this again to find a null index to store it. Hence

total number of operation is equal to twice the length of the array times some constant. Also size of array is some constant times input size n. That simply means O(n).

2. public int update (K key,T obj)
   Time complexity- O(n)
   Approach (proof)- I went iterating over array following (h1+ih2) till the time I find the key. When I find I overwrite the NodeDH<K,T> with K key, T obj and return i. Else when I don't find it I return -1 giving error. In worst case I have to go through entire array of size constant times n doing constant times n operations. Hence O(n).
3. public int delete (K key,T obj)
   Time complexity- O(n)
   Approach (proof)- Similar to that of update. Only difference is that instead of overwriting the node it made it null.
4. public boolean contains (K key)
   Time complexity- O(n)
   Approach (proof)- Same to that of delete, instead of making node null, here I left it unchanged and return true when it finds it and false when I don't find it.
5. public T get (K key)
   Time complexity- O(n)
   Approach (proof)- Same to that of contains, instead of return true when found I returned value of that key. In other word value of field T in NodeDH<K,T>. When not found throw new NotFoundException()
6. public String address (K key)
   Time complexity- O(n)
   Approach (proof)- same as that of get, instead of returning the value of field T in NodeDH<K,T> here it will return hashed value i.e. (h1+ih2)%hashtableSize in String form. When not found again throw new NotFoundException().

Note: I have implemented a generic hashtable of K,T. Any class having compareTo method can be passed to K. In other word it should implement comparable interface. I have not used any single type casting. My programme is fully generic.

MyHashTableSCBST<K entends comparable<K>,T>

Again this class is started with a array of NodeSCBST<K,T> which size is determined by the valued received from the constructor.

NodeSCBST<K,T> is a class with four field-K, T, NodeSCBST<K,T> and again NodeSCBST<K,T>. Constructor initializes these. Four different methods will

return these value respectively. Also other four more method will over write these values respectively.
It also has h1 method. This class also define following six abstract method of MyHashTable_<K,T> interface.

1. public int insert (K key,T obj)
   Time complexity- O(n)
   Approach (proof)- I hash the key and get the index for insertion in the array. If it is null at that index then make a NodeSCBST<K,T> and initialize its constructor with K key, T obj, left_node=null and right_node=null and insert this NodeSCBST<K,T> at this null index and return 1. This is basically our root node of BST. If it is not null then transverse through the BST. Go left of root node if it is less or go right if it it more than root node. Similarly do this for subsequent nodes and ultimately you will get the position when there exist no node. This is the required place. Insert your NodeSCBST<K,T> here and return nuber of nodes touched during this process. In Worst case it will be the height of BST which is further number of nodes in worst case which is again further number of total entries in hashtable (input size). Hence O(n). Note this will return -1 when key already exist in BST.
2. public int update (K key,T obj)
   Time complexity- O(n)
   Approach (proof)- Hash the key to get index number of the array. Go to that index number and you will find a node that is root of BST. If not found, that means that key doesnot exist in our hashtable hence error return -1. Else transverse through the BST to find the required key. Already mentioned in above proof of insert, how to transverse. If not found return -1 else return number of nodes touched so far and update the T field of NodeSCBST<K,T>. In worst case operations required will be height of tree, which is further number of nodes which is further input size n. Hence Complexity is O(n).
3. public int delete (K key)
   Time complexity- O(n)
   Approach (proof)- Same approach of update to find the node to delete. Then 3 cases-
   i) it has no child- give its parent node this reference null. Return nodes touched so far.
   ii) It has only one child- give its parent node this reference as child node. Return nodes touched so far.
   iii) It has both child- Search for its successor that is left most node in right most sub tree. Then replace this node with its successor and

then delete successor (which will use case (i) or (ii) depending upon availability of right child of successor). Return node touched so far. If what is to be deleted is not there than return -1.

Again in worst case all inputs in same index and BST forms a path and hence height of BST becomes input size and worst case operations become input size. Hence O(n).

4. public Boolean contains (K key)
   Time complexity- O(n)
   Approach (proof)- Same approach of update but here not overwriting will be done just return true if found and false if not found.
5. public T get (K key)
   Time complexity- O(n)
   Approach (proof)- Again find the key and return the value of field T of NodeSCBST<K,T>. throw new NotFoundException() if not found.
6. public String address (K key,T obj)
   Time complexity- O(n)
   Approach (proof)-Find the key and return the hash value i.e. h1 append "-" append sequence of 'L' or 'R' depending upon how you tranversed in finding the key. If not found throw new NotFoundException()

Note: Again my implementation is completely generic with no use of type casting. Key should extends comparable.

Student class will be passed for T obj. This has properly implemented Student_ interface. It has an additional method public String toString() which print all its value of fields on console in single line with same between then in desired order.

Pair<F,S> implements comparable<Pair<F,S>> because we will pass it for key. toString() gives a string value that is concatenation of (value of F).toString() and (value of S).toString(). It define compareTo() method- compare value of F and return 1 or -1 depending upon greater or lesser and in case of equal check for value at second if that is also same return 0 else return 1.

Assignment3.java Use fileReader and bufferedReader to read the file. Make Pair<String,String> as key and Student as obj after reading query line. Ofcourse we will make MyHashTableDH<Pair<String,String>,Student> or MyHashTableSCBST<Pair<String,String>,Student> depending upon command line argument "DH" or "SCBST".

Assignment fully completed. All methods are working as verified by me.