1] Sperate Chaining using Binary Search Tree

Let n be number of nodes present in the tree.

Operations :-

1) Insert :- insert function of SCBST will first calculate the hash value of the hashtable according to concatenation of first name and last name. It goes the that particular index value and traverses till the end of the tree (present in that location/index)and inserts and element at the end according to the result obtained after compairing first name of the new student with first name of all the students already present in the list.

Traversing time in the tree will be of order O(h) where h is the current height/depth of the tree. Insertion time is going be constant =O(1).

T(n)= O(h)

In binary tree, height of the tree is on an average equal to log(n).

Hence, T(n)=O(h)=O(log(n))

2) update :- undate function of SCBST will first calculate the hash value of the hashtable according to concatenation of first name and last name. It goes the that perticular index value and traverses the tree present at that location. It uses search() function of BST to search for the given student I'm the tree. This takes O(h) time (worst case) and hence O(logn) time. Once the element is found, updating it takes constant time O(1).

Therefore T(n)=O(logn)

3) delete:- delete function of SCBST will first calculate the hash value of the hashtable according to concatenation of first name and last name. It then goes to that particular index value and traverses though the tree present at that location. Is uses the search function to locate the node to be deleted. As mentioned above this take O(logn) time.

After locating the root, deletion takes O(1) time. After deleting the element, tree is updated such that the deleted node is replaced by the smallest node greater than the deleted one. This takes time less that logn since this time whole tree is not traversed but the remaining part is. Hence overall time complexity comes out to be O(logn)

T(n)=O(logn)

4) contains :- contains function of SCBST calls the search function and searches for the given element in the tree. Worst case complexity is O(h) where h is the height of the tree which is of O(logn)

Therefore search time complexity =O(logn)

T(n)= O(logn)

5) get():- Get function returns the object of the given key from the hashtable. It searches for the given object in the tree which takes O(logn) time. Once the element is found , returning it's object takes O(1) time.

T(n)= O(logn)

6) address :- time taken by address is the same as that taken by search() of the tree. It searches for the student in the tree and returns the sequence of left and right turns taken to reach the node.

Hence T(n)= O(logn)

2] Double Hashing

Operations :-

Double Hashing uses a hash function to calculate the hash index value for a given key using concatenation of first and last name. If we get same hash value for 2 keys, a new hash value is calculated using the formula and second key is sent there.

In double hashing if number of elements are N then our hashtable size is going to be between 1.5N and 2N (given) also we know that our hashtable size is always going to be a prime number. These two conditions reduces the probability of a collision to a great extent. Hence most of the times when an index value is calculated for a key it's unique and collision selmon happens. Hence when we have a specific element to be accessed in the hashtable, we calculate hash index value corresponding to that element and on an average , average time complexity of accessing that particular element in the hashtable is usually O(1).

1) insert :- insert function will calculate the hash value (O(1) time) and store the object of the student in the hashtable at that hash value. As mentioned above this takes O(1) time.

Therefore T(n)=O(1)

2) contains :- contains () function traverses through the hashtable by calculating hash value until we find the given key of the student in hash table.

T(n)=O(1)

3) update:- update function uses search function to contains() for the given key in the hashtable. Once that student object is found it takes O(1) time to update it with new information .

T(n)= O(1)

4) delete:- delete function looks for the given key in the hash table again, using contains() function. Once the object is found it's removed from the hashtable which takes O(1) time.

T(n)=O(1)

5) get():- function get() prints details of the given student if it's present in the hashtable. It again uses contains() function to search for the given key in the hashtable . Once found , it prints the details in O(1) time.

T(n)=O(1)

6) address ():- address function prints the hash index value calculated through the hash function. It is done using the formula and takes O(1) time.

T(n)=O(1)