**CS 5V81.012 – Special Topics in Computer Science – F16**

**Optional Project 7**

**Team Members:**

**Alwin Johns(axj155830)**

**Divya Vudem(dxv151430)**

**Manoj Kumar Ellanti(mxe150630)**

**Srabonti Chakraborty(sxc154030)**

**Question 1:**

**Remove\_Duplicates**.**java** has the code to this solution. The given code contains array list of objects from class Classeh.java and Hash Set is used to remove duplicates from the list.

SECOND Approach:

)HashSet to remove duplicates is implemented using **RemoveDuplicate.java**. In this method, DummyObject is created to make note of index{position in the array}, count and value of element. increment() method is used to just increment the count. Hashcode() is implemented which return the name.hashcode() java implementation of the hashcode().

Similarly, equals method is implemented to compare two objects.

New Elements are added to hashset and hashmap if they are not there previously. For every other element if it is there in the set, hashmap gives its previous index in the array, make both to null.

Continue this till the end of array. Once, End is reached HashMap contains all the indexes of the distinct elements. iterate through array from 0 and start replceing the elements using hashmap's positions.

**Question 2:**

MostFrequentlUsingHashMap.java is used to implement the most frequently appearing element in the array.

Count object is created with index and count as its attributes and increment method.

HashMap is created<T,Count> this contains the key as input array element. Look if the T exists in the Hashmap if so update its count and look for updations in the max and maxat.

**Question 3:**

**SeparateChaining.java** has the code to this solution where separate chaining hashing is implemented using array of linked lists.

**TwoChoice2.java** is used to implement the code for the implementation of Two choice hashing. Here Simple comparisions between minimum of arraylist elements of are made between two hashpositions in the table.

**Question 4:**

Hopscotch hashing is implemented using **hopscotchhash.java**.

ADD(), hop is maximum gap taken

An element is added to the table , if there is an empty place to it within the s0+hop. Everytime the filled elements hashcodes are noted in arrayList to look for a potential replacement to it.

the element is added if an null place is found or the element is found to be null(in case of deletion only the element is made null for indication of previous occupancy).

Line 92 looks for a potential replacement if needed using the arraylist populated.

get(), remove(), contains()

All the above are based on almost same code where u are going to look for the element within S0+hop, In remove once the object is found , we will just make its element null.

**DoubleHashing.java** is used to implement the code for the double hashing here two hashfunctions are used and, ProbeSequence is calculated for every add using the Hash functions.

get(), remove(), add(), contains () are implemented similar to Hopscotch if not for the hoping concept.

**Question 5:**

**TreeMap\_Comparisions.java** file has code to compare the performances of Tree Map, Hash Map, Linked List and Skip List using functions add, remove and contains.

Below are the performances on different sizes of elements

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Size | Tree Map | Hash Map | Linked List | Skip List |
| 1000,000(one million) | Time: 2678 msec.  Memory: 27 MB / 123 MB. | Time: 810 msec.  Memory: 34 MB / 123 MB. | Time: 708 msec.  Memory: 44 MB / 123 MB. | Time: 13855 msec.  Memory: 50 MB / 155 MB. |
| 1000,0000(ten  Million) | Time: 45107 msec.  Memory: 177 MB / 254 MB. | Time: 16848 msec.  Memory: 194 MB / 350 MB. | Time: 23798 msec.  Memory: 265 MB / 601 MB. | Time: 567855 msec.  Memory: 1298 MB / 786 MB. |
| 2000,0000 (Twenty  Million) | Time: 81802 msec.  Memory: 318 MB / 467 MB. | Time: 25860 msec.  Memory: 408 MB / 682 MB. | Time: 36575 msec.  Memory: 680 MB / 955 MB. | Time: 1781423 msec.  Memory: 6431 MB / 978 MB. |