**PROJECT-3 REPORT**

**(Implementation of Multi-dimensional search to the Customer data base to maximize the efficiency of customer information retrieval and updates)**

**Implementation details:**

To accomplish the customer retrieval and update functions, following 4 composite data structures have been created, inserted with appropriate data, updated and deleted the entries within this data structures whenever required. Also, if changes are performed on one data structure, corresponding updates are made in all the other data structures.

1. Hash map with Customer ID as key and Customer object as value: (**cust** data structure in the program)

**HashMap<Customer ID, Customer Object>**

**About this composite data structure:**

* Populated the customer with its ID as key and Customer object (with ID specified, TreeSet of categories provided, and amount to 0) in the **insert** function.
* Deleted the customer’s entry associated with the customer ID provided, from this data structure in the **delete** function.
* Update the customer list in this TreeSet data structure associated with the customer object in **addinterests** and **removeinterests** functions.
* Update the revenue of the customer in the corresponding customer object in **addrevenue** function.
* Use this data structure in **find** function to find the customer using his/her ID and **TopThree** to get the amount(along with cents) of top 3 customers.

1. Hash map with category list as key with value as a Tree map of amounts (amount\*100) associated with that category. The customers associated with a particular tree map node (amount) are maintained as list against that tree map node. (**catcust** data structure in the program).

**HashMap<Category,TreeMap<(Amount\*100),LinkedList<Customer ID’s>>>**

**About this composite data structure:**

* Add the customers to the linked list of amount node ‘0’ along each of his/her interested category in this hashmap during **insert** operation.
* Delete the customers from the linked list of amount (amount\*100) node corresponding to them in every treemap of his/her category of interests during **delete** operation.
* Use this data structure to find the top three customers for the category provided as the key to this hashmap.
* Update the data structure with the customer ID in the list of customers against each categories newly added/ removed for this customer during **addinterests** and **removeinterests** functions.

1. Tree map of amounts with list of customers associated with a particular amount are maintained as value against that tree map bode. (**amtcust** data structure in the program)

**TreeMap<(Amount\*100),LinkedList<Customer ID’s>>**

**About this composite data structure:**

* Add customer ID to the list across amount key ‘0’ in this treemap during **insert** operation.
* Delete the customer ID from the list against the key as the current amount (amount\*100) of the customer in the treemap during **delete** operation.
* Reposition the customer ID at the list maintained against the exact current amount key of the customer in the amtcust data structure, after the **addrevenue** operation**.**
* Use this data structure in range function to determine the customers with amounts in the given range.

1. Hash map with tree set of category lists as key and list of customers as value whose category list exactly matches with the tree set of categories maintained in the key.

**HashMap<TreeSet<Categories>,LinkedList<Customer ID’s>>**

**About this composite data structure:**

* With the categories treeset of the new customer added as the key, add the customer ID as the value across this key in this data structure during **insert** operation.
* Delete the key with the treeset of the customer categories whenever the customer is deleted during **delete** operation (deleted exactly if no customer is present in the customer list maintained under the value of the treeset of customer categories key).
* Reposition the customer ID at the list maintained against the key from the old category list as key to the new category list as key. Delete the old key if no customer associated with this category list. This operation occurs in both **addinterests** and **removeinterests** operations.
* Use the data structure to determine the number of customers who have atleast 5 set of categories with at least one another customer having exactly the same set of interests.

**Test procedure:**

1. Open CustomerMaintain.java
2. Go to ‘Run configurations’->arguments. Provide the file name containing the test data (with test data containing all the function calls to the implemented functionalities of customer database) to run the program in the program arguments. Place this file in the same directory in which the ‘src’ folder of above programs were present.
3. Click on ‘run’ and press OK.
4. The expected output after calling all the function calls present in the file is displayed along with the running time of the entire process along with the memory space occupied to accomplish these function calls.
5. Repeat above steps for the different test data present in different files (change the file name in the Program arguments and repeat the above test procedure).

**Test results:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Folder: p3-in-degen** | | | |
| **Test data file name** | **Result** | **Running time**  **(in millisec)** | **Space occupied (Memory used/Memory available)** |
| in1.txt | 38 | 12 msec | 13 MB/128 MB |
| inc.txt | 989 | 124 msec | 24 MB/128 MB |
| ink.txt | 419 | 424 msec | 70 MB/163 MB |
| inl.txt | 510 | 36357 msec | 1258 MB/1689 MB |
| inxk.txt | 348 | 3700 msec | 169 MB/737 MB |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Folder: p3-s2** | | | |
| **Test data file name** | **Result** | **Running time**  **(in millisec)** | **Space occupied**  **(Memory used/Memory available)** |
| p3-s2-ck.txt | 282 | 27767 msec | 975 MB/1239 MB |
| p3-s2-d.txt | 802 | 208 msec | 18 MB/128 MB |
| p3-s2-k.txt | 264 | 293 msec | 40 MB/128 MB |
| p3-s2-l.txt | 249 | 64 msec | 15 MB/128 MB |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Folder: p3-s3** | | | |
| **Test data file name** | **Result** | **Running time**  **(in millisec)** | **Space occupied**  **(Memory used/Memory available)** |
| p3-s3-ck.txt | 705 | 27630 msec | 942 MB/1213 MB |
| p3-s3-d.txt | 861 | 196 msec | 18 MB/128 MB |
| p3-s3-k.txt | 990 | 300 msec | 41 MB/128 MB |
| p3-s3-l.txt | 558 | 60 msec | 15 MB/128 MB |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Folder: p3-s4** | | | |
| **Test data file name** | **Result** | **Running time**  **(in millisec)** | **Space occupied**  **(Memory used/Memory available)** |
| ss-ck.txt | 682 | 37158 msec | 1205 MB/1687 MB |
| ss-xk.txt | 738 | 3516 msec | 148 MB/530 MB |

**Test for samesame operation:**

**Testdata:**

Insert 801 101 102 103 104 0

Insert 802 101 102 103 104 0

Insert 803 101 102 103 104 105 0

Insert 804 101 102 103 104 105 0

Insert 805 101 102 103 104 105 0

Insert 901 201 202 203 204 205 206 0

Insert 902 201 202 203 204 205 206 207 0

Insert 903 201 202 203 204 205 206 207 208 0

SameSame

AddInterests 803 106 0

SameSame

AddInterests 901 207 0

SameSame

RemoveInterests 903 208 0

SameSame

End

|  |  |  |
| --- | --- | --- |
| **Result** | **Running time**  **(in millisec)** | **Space occupied** |
| 31 | 2512 msec | 13 MB/ 128 MB |