CS 5V81.001: Implementation of data structures and algorithms

Project 3

Akshay Thakare (ast140230)

Description: Efficient use of various data structures for implementation of various functions.

Problem Statement: Multi-dimensional search: Consider a retailer like Amazon. They carry tens of thousands of products, and each product has many attributes (Name, Size, Description, Keywords, Manufacturer, Price, etc.). Their web site allows users to specify attributes of products that they are seeking, and displays products that have most of those attributes. To make search efficient, the data is organized using appropriate data structures, such as balanced trees and hashing. But, if products are organized by Name, how can search by price be implemented efficiently? The solution, called indexing in databases, is to create a new set of references to the objects for each search field, and organize them to implement search operations on that field efficiently. As the objects change, these access structures have to be kept consistent.

In this project, each customer object has 3 attributes: id (long int), categories (one or more integers 1-999), and amount (dollars and cents). The amount field stores the total revenue that the customer has generated for our company. The categories field is a set of department codes of our company (encoded as an integer in [1,999]) that stock items of interest to the customer. Customers are uniquely identified by their id (key field).

Following are the functions implemented in this project:

1. Insert(id, categories): add a new customer (with amount = 0) who is interested in the given set of categories. Returns 1 if the operation was successful, and -1 if there is already another customer with the same id (in which case no changes are made).

2. Find(id): return amount spent by customer until now (or -1, if such a customer does not exist).

3. Delete(id): delete customer's records from storage. Returns the amount field of the deleted customer (or -1, if such an id did not exist).

4. TopThree(k): given a category k, find the top three customers (in terms of amount spent) who are interested in category k. Returns the sum of the amounts of the top three customers, truncated to just dollars.

5. AddInterests(id,categories): Add new interests to the list of a customer's categories. Some of them may already be in the list of categories of this customer. Return the number of new categories added to that customer's record. Return -1 if no such customer exists.

6. RemoveInterests(id, categories): Remove some categories from the list of categories associated with a customer. Return the number of categories left in the customer's record. It is possible that the user has no categories of interest after this step (if all his categories of interest are removed). Return -1 if no such customer exists.

7. AddRevenue(id, purchase): update customer record by adding a purchase amount spent by a customer on our company's product. Returns the net amount spent by the customer after adding this purchase, truncated to just dollars (-1 if no such customer exists).

8. Range(low, high): number of customers whose amount is at least "low" and at most "high".

9. SameSame(): Find customers who have exactly the same set of 5 or more categories of interest.

10. NumberPurchases(id): The number of times customer has purchased products. Return -1 if no such customer exists.

Classes in the Project:

1. **Customer:** The Customer class is used to create customer object with attributes: id, amount, categories, number of purchases made by the customer.

2. **MultiDSearch**: The MultiDSearch class implements functions insert, find ,delete, topthree, addinterest, removeinterest, addrevenue, range, samesame, numberpurchases.

Data Structures Used:

1. Hashmap<Long,Customer> hCustomer: This data structure is used to store the customers with id as the key and Customer object as value so that the customers can be searched in O(1) time.

2. Hashmap<Integer,TreeSet<Customer> hCategoryAmount: This data structure is used to store the category as the key and value as the customer object sorted in the order of its amount attribute. This is used for TopThree function so that it can search the category in O(1) time.

3. TreeSet<Customer> tCustomer: This data structure is used to store the customers in the order of the amount. This is used in the Range function to find the number of the customers in a specific range of the amount.

4. BitSet bCategories: This data structure is used to store the categories in which the customer is interested. As indexing is supported by it, we can find the number of the categories the customer is interested by using in-built function such as 'cardinality()' which gives the current number of bits set in the BitSet.

Results:

Sr. No.	Input File	Output	Time in ms	Memory
1.	in1.txt	38	16	2 MB / 128 MB
2.	inc.txt	989	78	11 MB / 128 MB
3.	ink.txt	419	313	21 MB / 128 MB
4.	inl.txt	510	22931	526 MB / 792 MB
5.	Inxk.txt	348	1546	66 MB / 365 MB
6.	p3-s2-ck.txt	282	14939	204 MB / 768 MB
7.	p3-s2-d.txt	802	156	21 MB / 128 MB
8.	p3-s2-k.txt	264	219	5 MB / 128 MB
9.	p3-s2-l.txt	93	62	4 MB / 128 MB
10.	p3-s3-ck.txt	336	14420	217 MB / 768 MB
11.	p3-s3-d.txt	7	164	20 MB / 128 MB
12.	p3-s3-k.txt	990	243	6 MB / 128 MB
13.	p3-s3-l.txt	725	63	4 MB / 128 MB
14.	ss-ck.txt	682	28547	402 MB / 592 MB
15.	ss-xk.txt	738	2144	118 MB / 364 MB