Vaishakhi Kulkarni Net Id:vpk140230

PROJECT REPORT

Abstract:

Implemented a TreeMap data structure which has key as Customer Id and value as the whole object of that customer to store information related with id, categories and amount. This is a main data structure which is used in various operations. Also implemented HashMap to perform TopThree operation. In this the key is a integer of category and the value is a TreeSet which contains objects of various customer who has that category included in it. Used these two data structure have implemented various operations specified below.

Problem Statement:

Perform multi-dimensional search on each customer object which has 3 attributes:
Id (long int)
Categories (one or more integers 1-999)
Amount (dollars and cents)
Perform following operations:

1. Insert (id, categories)
2. Find (id)
3. Delete (id)
4. Top Three (k)
5. Add Interests (id, categories)
6. Remove Interests (id, categories)
7. Add Revenue (id, purchase)
8. Range (low, high)
9. Same Same
10. Number Purchases (id)

Methodology:

Created a class **cust** of customer. This class has attributes like Id, categories, amount, count and seen. As every customer ca have many categories, the categories are stored in HashSet. Implemented as TreeMap **customer** which has key as Customer ID and value as the object of that corresponding customer. This data structure in used in various operations. Also created a HashMap **topThree** which stores category number as key while implemented a TreeSet of customers which has that category number in it. This data structure is used to perform TopThree operations. Every time any updates are performed must be updated in these two data structures. By using HashSet, TreeMap and HashMap we can dynamically add records and the time to perform this operations is also very less.

Primary Functions implemented are:

1. Insert (id, categories): Adds new customer who is interested in the given set of categories, in **customer** data structure and should **return 1** else should **return -1**. Also insertion should be done in **topThree** data structure.

2. Find (id): Search in **customer** data structure whether the id is present. If it is resent it will return total amount spent by the customer until now otherwise will **return -1**.

3. Delete (id): Search in **customer** data structure whether the id is present. If it is present it will return the amount field of the deleted customer. Else will **return -1**. As we delete the record from **customer** we should also makes corresponding changes in **topThree** data structure.

4. TopThree (k): Given is a category k. To find top three customers who is interested in category k we will use **topThree** data structure to perform this task. Should calculate the total amount of top three customers and return that amount which is truncated in dollar.

5. AddInterests (id, categories): Make changes in **customer** and **topThree** data structure. First we check whether the customer id is present or not. If it is not present **return -1** otherwise as the new categories to the existing category list of that customer id. If the category already exist we should not add in the category list of that customer. Make updates on both data structure **return the number of new categories added to the customer records**.

6. RemoveInterest (id, categories): Make changes in **customer** and **topThree** data structure. First we check whether the customer id is present or not. If it is not present **return -1** otherwise remove some categories from the list associated with a customer. **Return the numbers of categories left** in the customer records. While performing deletion of category and if a customer has no category left, delete that record on both data structure.

7. AddRevenue (id, purchase): It checks whether the customer id is present or not. If it is not present **return -1** else it updates the customer record by adding purchase amount spent by a customer on company products. **Returns the net amount** of the customer after the purchase amount is included. This amount is updated in the records of **customer** and **top Three**. Whenever any updates are performed on the specified customer, the customer field count is incremented. This field is used in NumberPurchase operation.

8. Range (low, high): Check is **customer** data structure the number of customers who has amount in specified range. **Returns the number of those customers within the given range**.

9. SameSame(): Finds customers who have exactly same set of 5 or more categories of interest. For this used two loops. Other loop will traverse for whole records in the **customer** data structure and while the inner loop will also traverse for whole records except for the one which is present in the other loop. Then added a condition that **the size of the identical categories must be more than 5 and also the customer's field seen is 0**. If this condition is satisfied then **inner loop customer's seen field is set to 1** and counter is incremented. So by this mean we can remove the records which are identical. After completion of inner loop set **other customer's seen field as 1**. Continue until loop finishes. After all is done again set customer's field of all customers to 0.

10. NumberPurchase (id): It checks whether the customer is present in the **customer** data structure. If no **return -1** else it returns the value associated with field count of customer object **cust.count**.

Development Platform:

Operating System: Windows8

Hardware Specification: Intel CORE i5 processor, 8GB RAM

Development IDE: Eclipse

Test Cases:

a) **Following files have basic file operations provided in the data set:** Insert (id, categories),Find (id),Delete (id),Top Three (k),Add Interests (id, categories),Remove Interests (id, categories),Add Revenue (id, purchase)

1. In1.txt
2. Inc.txt
3. Ink.txt
4. Inl.txt

b) **Following files have basic file operations provided in the data set:** Insert (id, categories),Find (id),Delete (id),Top Three (k),Add Interests (id, categories),Remove Interests (id, categories),Add Revenue (id, purchase), Range (low, high), Number Purchases

1. p3-s2-ck.txt
2. p3-s2-d.txt
3. p3-s2-k.txt
4. p3-s2-l.txt
5. p3-s3-ck.txt
6. p3-s3-d.txt
7. p3-s3-k.txt
8. p3-s3-l.txt

c) **Following files have basic file operations provided in the data set:** Insert (id, categories),Find (id),Delete (id),Top Three (k),Add Interests (id, categories),Remove Interests (id, categories),Add Revenue (id, purchase), Range (low, high), Number Purchases(id), SameSame.

1. same1.txt
2. ss-xk.txt

Results:

Following table shows the performance of the data structures implemented to perform above operations.

Input File	Output	Time Taken	Space Utilized
in1.txt	38	15msec.	2 MB / 128 MB
inc.txt	989	118 msec.	12 MB / 128 MB.
ink.txt	419	438 msec.	36 MB / 163 MB.
inl.txt	510	16704 msec.	638 MB / 1030 MB.
p3-s2-ck.txt	282	17210 msec.	324 MB / 816 MB.
p3-s2-d.txt	802	229 msec.	21 MB / 128 MB.
p3-s2-k.txt	264	347 msec.	8 MB / 128 MB.
p3-s2-l.txt	93	109 msec.	4 MB / 128 MB.
p3-s3-ck.txt	705	21611 msec.	349 MB / 814 MB.
p3-s3-d.txt	861	224 msec.	21 MB / 128 MB.
p3-s3-k.txt	990	344 msec.	8 MB / 128 MB.
p3-s3-l.txt	558	71 msec.	4 MB / 128 MB.
same1.txt	31	15 msec.	2 MB / 128 MB.
ss-xk.txt	738	93366 msec.	146 MB / 377 MB.

Note: All the reading are calculated after taking average of 5 outputs.

Conclusion:

Using combinations of data structure we can reduce the runtime complexity to perform various operations on database. The combination of data structure must be selected as per the requirements. Proper use of data structure gives optimum results.

References:

Class notes - Balaji

Introduction of Algorithms - Thomas Cormen, Charles Leiserson, Ronald Rivest , Clifford Stein