Implementation of Multi-Dimensional Search

Submitted by **Group 12**
- Anirudh K V
- Malini K Bhaskaran
- Neha Nirmala Srinivas
- Saumya Ann George

Executive summary

This project implements the multi-dimensional search operation**.**

Multi-dimensional search:

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 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.

Problem Statement

In this project, each object has 3 attributes: id (long int), description (one or more long ints), and price (dollars and cents). The following operations are supported:

1. Insert(id,price,description): insert a new item. If an entry with the same id already exists, its description and price are replaced by the new values. If description is empty, then just the price is updated. Returns 1 if the item is new, and 0 otherwise.
2. Find(id): return price of item with given id (or 0, if not found).
3. Delete(id): delete item from storage. Returns the sum of the long ints that are in the description of the item deleted (or 0, if such an id did not exist).
4. FindMinPrice(n): given a long int n, find items whose description contains n (exact match with one of the long ints in the item's description), and return lowest price of those items.
5. FindMaxPrice(n): given a long int n, find items whose description contains n, and return highest price of those items.
6. FindPriceRange(n,low,high): given a long int n, find the number of items whose description contains n, and their prices fall within the given range, [low, high].
7. PriceHike(l,h,r): increase the price of every product, whose id is in the range [l,h], by r% Discard any fractional pennies in the new prices of items. Returns the sum of the net increases of the prices.
8. Range(low, high): number of items whose price is at least "low" and at most "high".
9. SameSame(): Find the number of items that satisfy all of the following conditions:

   The description of the item contains 8 or more numbers, and,

   The description of the item contains exactly the same set of numbers as another item.

Pseudocode

Insert method and Data structure used :

We have used tree Maps for storing the product information. Used 3 tree Maps as follows.

Tree Name	Types
idTree	TreeMap<Long,Product>
nameMap	TreeMap<Long, NameNode>
priceMap	TreeMap<Double, List<Long>>

2 main scenarios were covered in the insert function.

1. When the product is already present in the idTree

   If the product is present in idTree, we updated the nameMap and priceMap with the incoming values.

Old values are removed from the tree maps and new values inserted. Before updating nameMap checks were performed to make sure that the incoming product description is having non zero length.

2. When the product is not present in the idTree, then idTree , nameMap and priceMap have been updated with the new product details

Delete:

Obtain the product from the idTree. Remove the product from the NameNode description list. Remove the product from the idTree. Remove the product id from the priceMap.

PriceHike :

From the idTree we obtain all the ids which are in the specified range. The new price is calculated based on the old price and the rate specified. The product is removed from the priceMap based on the old price. The product is reinserted into the priceMap based on new price.

findPriceRange :

Obtain the list of products from the nameMap description list. If the price of the product is in the specified range increment count.

findMinPrice :

Obtain the NameNode from the nameMap for the specified description length. If such a NameNode exists obtain the product list. Iterate through the product list while keeping track of the product with minimum price.

findMaxPrice :

Obtain the NameNode from the nameMap for the specified description length. If such a NameNode exists obtain the product list. Iterate through the product list while keeping track of the product with maximum price.

SameSame :

Obtain products from idMap and add it to the product list. For all the products in the product list if the description list length is greater than 7, sort the description list and check if the description list exists in the HashMap descCountMap. If the description is not present make a new entry with initial frequency one else increment the frequency.

Test Results

Input File Name	**Output**
lp4-1.txt	1450.08
lp4-2.txt	4146.32
lp4-3-1k.txt	52252.36
lp4-4-5k.txt	490409.01
lp4-5-ck.txt	173819092858.22
lp4-bad.txt	1016105100

Discussion of Results

Input File Name	Output	Execution Time Excluding the time for reading input(ms)	Execution time including the time for reading input (ms)
lp4-1.txt	1450.08	6	89
lp4-2.txt	4146.32	6	72
lp4-3-1k.txt	52252.36	32	157
lp4-4-5k.txt	490409.01	154	450
lp4-5-ck.txt	173819092858.22	6995	8111
lp4-bad.txt	1016105100	3349	18279

Conclusion

We have implemented the multi dimensional search functionality using treeMaps and there by we understood how to combine data structures and use them to solve real world problems.