Multi-dimensional search Name : Sasidhar Evuru

Abstract - The main aim of the project is to use multiple Data Structures to perform efficient search using different attributes of a product such as cost, name.

Problem Statement – Perform operations like Insert, find, delete, find minimum price, find maximum price, find price range, price hike on a huge data efficiently which is stored in the form of different data structures.

Implementation - Data is stored using different Data structures. The implementation is done as follows:

1. We have a hashmap (H1) with key as id and Product (object) as value. We store a new product in this hashmap so that access can be done in O (1) time.
2. The name part of the product contains multiple long integers and since many operations such as search are performed on all the objects having this name part, we are saving all the product references with same long integer part in their name under one roof. So we need one more hashmap (H2) which should have key as the long integer (part of the name) and value as a Redblack tree which store all the Product object references in the form of a balanced binary search tree. In the Redblack tree the products were stored with cost a the key value , if the product value is same then the product is arranged in the tree based upon the ID value.
3. We have a separate Redblack tree (RB1) with the Id as the key to Red Black tree because we have operations that require all the Ids in the particular range. To make these operations efficient it's better to have a separate Redblack tree with ID.

Functions implemented :

1. Insert (id, price, name) – This creates a new Product object with the given attributes and checks for whether there is a product with the same id in the H1. If there is already a product with the same Id, for each individual component of the name of the product delete the product reference in the Redblack tree of the Hashmap (H2), then delete the id entry from the hash map H1, and then the newly created object is inserted first in H1, and then for each name part in H2. Otherwise if it's the first time insertion same order is followed like insertion in H1, and then for each name part in H2 and in addition the id is added to Redblack tree RB1.
2. Find (id) - This simply finds the cost from the H1 for the corresponding product with the given id.
3. Delete(id)- For each individual component of the name of the product delete the product reference in the Redblack tree of the Hashmap (H2), then delete the id entry from the hash map H1, the delete the id from Redblack tree RB1.

4. FindMinPrice (n) – From H2 find the Redblack tree with the given part of the name and return minimum price from the Redblack tree.
5. FindMaxPrice (n) - From H2 find the Redblack tree with the given part of the name and return maximum price from the Redblack tree.
6. FindPriceRange(n,low,high) – From H2 find the Redblack tree with the given part of the name and return number of items with the cost between the given range.
7. PriceHike (l, h, r) – From RB1 get all the ids in the given range and increment their prices by the given percent.

Test results – The test cases are all tested with the given inputs and got correct output for all the cases. The test case with the one lakh input ran in about 13 seconds and got the correct result**.**

References :

1. For RebBlackBST.java
   http://algs4.cs.princeton.edu/33balanced/RedBlackBST.java.html

2. For Qeue.java
   http://algs4.cs.princeton.edu/13stacks/Queue.java