**Implementation of Advanced Data Structures and Algorithms**

**Project - 6**

**Suraj Sangani(sns140230)**

**Abstract:**

Consider the web site of a seller like Amazon. They carry tens of thousands of products, and each product has many attributes (Name, Size, Description, Keywords, Manufacturer, Price, etc.). The search engine allows users to specify attributes of products that they are seeking, and shows products that have most of those attributes. To make search efficient, the data is organized using appropriate data structures, such as balanced trees. 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:**

The aim of this project is to implement Multi-Dimensional Search. In this project, each object has 3 attributes: id, name, and price. The following operations are supported:

a. Insert(id,price,name): insert a new item. If an entry with the same id already exists, its name and price are replaced by the new values. If name is empty, then just the price is updated. Returns 1 if the item is new, and 0 otherwise.

b. Find(id): return price of item with given id (or 0, if not found).

c. Delete(id): delete item from storage. Returns the sum of the long ints that are in the name of the item deleted (or 0, if such an id did not exist).

d. FindMinPrice(n): given a long int n, find items whose name contains n (exact match with one of the long ints in the item's name), and return lowest price of those items.

e. FindMaxPrice(n): given a long int n, find items whose name contains n, and return highest price of those items.

f. FindPriceRange(n,low,high): given a long int n, find the number of items whose name contains n, and their prices fall within the given range, [low, high].

g. 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.

**Development Platform:**

Eclipse Java EE IDE Luna 4.4.0;

Windows 7, 8.00GB RAM, 64-bit OS

**Methodology:**

The project utilizes an AVLTree in conjunction with a HashMap. This enables to sort the items efficiently. The AVLTree stores all the nodes sorted via id. The HashMap contains the key as the id and the reference to the node as the value for the key.

* The user types in the input.
* The input is read using scanner. The scanner scans the operation entered by the user.
* Depending on the input entered by the user, the project performs the required operation.
* If the user enters ‘Insert’ keyword, the id, price and name are stored and the insert operation is called.
* If the user enters ‘Find’ keyword, the project performs a find operation and returns the price of the item to be found.
* If the user enters ‘Delete’ keyword, the remove function is called and the sum of the log ints of the names are returned and stored in the result.
* If the user enters ‘FindMinPrice’ keyword, the minPrice function is called which returns the minimum price of the item stored for the given name.
* If the user enters ‘FindMaxPrice’ keyword, the maxPrice function is called which returns the maximum price of the item stored for the given name.
* If the user enters ‘FindPriceRange’ keyword, the name, low and high values are stored and the pricerange operation is called.
* If the user enters ‘PriceHike’ keyword, the low, high and r(% increase) values are stored and the pricehike operation is called.
* If the input starts with a pound ‘#’, the rest of the statement is considered as a comment and ignored by the project.
* If the user enters ‘exit’ keyword, the programs exits while returning the sum of values of all the operations called.

**Test Cases:**

1) *Input:*

Insert 22 19.97 475 1238 9742 0

Insert 12 96.92 44 109 0

Insert 37 47.44 109 475 694 88 0

PriceHike 10 22 10

FindMaxPrice 475

Delete 37

FindMaxPrice 475

*Output:*

1450.08

2) *Input:*

Insert 22 19.97 475 1238 9742 0

Find 22

Insert 22 29.97 0

Find 22

Insert 12 96.92 44 109 0

Insert 37 47.44 109 475 694 88 0

FindMinPrice 475

Insert 22 29.97 998 997 996 995 0

Find 22

FindMinPrice 475

Delete 22

Find 22

*Output:*

4146.32

**Assumptions:**

* All operations begin with the keyword specified in the input specification. Program will terminate for invalid input.

**Discussion** **of** **Results:**

The project successfully implements the Multi-Dimensional Search using an AVLTree and a HashMap. The items are sorted via id and can be retrieved in a sorted price order. Also such a program performs useful operations which are needed in real life such as searching for maximum price or minimum price of an item, and searching for an item with given name in a specific price range.

**Conclusion:**

The program successfully makes search efficient and implements the multi-dimensional search via use of AVL Trees and HashMaps in association with each other.