GROUP 05 - LONG PROJECT 4
MULTIDIMENSIONAL SEARCH

Introduction:

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, 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.

Functions Implemented:

- 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.

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

- 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).

- 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.

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

- 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].

- 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.

- Range(low, high): number of items whose price is at least "low" and at most “high".

- 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.

DEVELOPMENT SETUP:

Java SE 1.8
Eclipse IDE

TEST RESULTS:

Input File: lp4-data/lp4-1.txt

```
1450.08
Time: 27 msec.
Memory: 4 MB / 128 MB.
```

Input File: lp4-data/samesame.txt

```
22.00
Time: 31 msec.
Memory: 4 MB / 128 MB.
```

Input File: lp4-data2/lp4-bad.txt

```
1016105100.00
Time: 14621 msec.
Memory: 176 MB / 865 MB.
```

Input File: lp4-data/lp4-2.txt

```
4163.79
Time: 33 msec.
Memory: 4 MB / 128 MB.
```

Input File: lp4-data/lp4-3-1k.txt

```
60413.74
Time: 103 msec.
Memory: 8 MB / 128 MB.
```

Input File: lp4-data/lp4-4-5k.txt

```
518481.61
Time: 203 msec.
Memory: 25 MB / 128 MB.
```

Input File: lp4-data/lp4-5-ck.txt

```
179923885670.44
Time: 1372 msec.
Memory: 90 MB / 231 MB.
```

DISCUSSION OF RESULTS:

We have managed to get the results close to the test output, barring some round off errors, which maybe due to the double arithmetic which has been performed.

Also, the running time of the input has been close to few milliseconds

For the bad input, the code runs at 14 seconds.

CONCLUSION:

The program for Multidimensional Search is implemented and the output and running times for various inputs is displayed. It is found that the usage of TreeMap and HashMap effectively has reduced the running times of the program.

REFERENCES:

Java Documentation on TreeMap, HashMap, NavigableMap and Iterator for HashMaps