```
CS 6301.002.  Implementation of advanced data structures and algorithms
Spring 2016;  Fri, Mar 18.
Long Project 4: Multi-dimensional search

Ver 1.0: Initial description (Fri, Mar 18: 5:30 PM).
Ver 1.1: Modified starter code.  Download again (Mon, Mar 21, 7:30 AM).
Max excellence credits: 1.0.
Due: 11:59 PM, Sun, Apr 10 (1st deadline), Sun, Apr 24 (2nd deadline).
```

Code base

Java library, java examples discussed in class by instructor. Do not use code from outside sources.
Starter code: MDS.java, LP4Driver.java.

Project Description

Multi-dimensional search: 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.

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.

   Creative solutions that are elegant and efficient will be awarded excellence credit. This operation returns the number of items that satisfy both conditions.

Implement the operations using data structures that are best suited for the problem. It is recommended that you use the data structures from Java's library when possible.

Input specification

Initially, the store is empty, and there are no items. The input contains a sequence of lines (use test sets with millions of lines). Lines starting with "#" are comments. Other lines have one operation per line: name of the operation, followed by parameters needed for that operation (separated by spaces). Lines with Insert operation will have a "0" at the end, that is not part of the description.

```
Sample input:
Insert 22 19.97 475 1238 9742 0
# New item with id=22, price="$19.97", description="475 1238 9742"
# Return: 1
#
Insert 12 96.92 44 109 0
# Second item with id=12, price="96.92", description="44 109"
# Return: 1
#
Insert 37 47.44 109 475 694 88 0
# Another item with id=37, price="47.44", description="109 475 694 88"
# Return: 1
#
PriceHike 10 22 10
# 10% price increase for id=12 and id=22
# New price of 12: 106.61, Old price = 96.92.  Net increase = 9.69
# New price of 22: 21.96.  Old price = 19.97.  Net increase = 1.99
# Return: 11.68  (sum of 9.69 and 1.99)
#
FindMaxPrice 475
# Return: 47.44 (id of items considered: 22, 37)
#
Delete 37
# Return: 1366 (=109+475+694+88)
#
FindMaxPrice 475
# Return: 21.96 (id of items considered: 22)
#
End
# Lines after "End" are not processed.
```

Output specification

The output is a single number, which is the sum of the following values obtained by the algorithm as it processes the input.

```
Output:
1450.08
```