**Price Check**

**QUESTION DESCRIPTION**

There is a shop with old-style cash registers. Rather than scanning items and pulling the price from a

database, the price of each item is typed in manually. This method sometimes leads to errors. Given a list of items and their correct prices, compare the prices to those entered when each item was sold. Determine the number of errors in selling prices.

Example

products = ['eggs', 'milk', 'cheese']

productPrices = [2.89, 3.29, 5.79]

productSold = ['eggs', 'eggs', 'cheese', 'milk']

soldPrice = [2.89, 2.99, 5.97, 3.29].

Price

|  |  |  |  |
| --- | --- | --- | --- |
| Product | Actual | Expected | Error |
| eggs | 2.89 | 2.89 |  |
| eggs | 2.99 | 2.89 | 1 |
| cheese | 5.97 | 5.79 | 1 |
| milk | 3.29 | 3.29 |  |
|  |  |  |  |
|  |  |  |  |

The second sale of eggs has a wrong price, as does the sale of cheese. There are 2 errors in pricing.

**Function Description**

Complete the function priceCheck in the editor below.

priceCheck has the following parameter(s):

string products[n]: each products[i] is the name of an item for sale

string productPrices[n]: each productPrices[i] is the price of products[i]

string productSold[m]: each productSold[j] is the name of a product sold

float soldPrice[m]: each soldPrice[j] contains the sale price recorded for productSold[j].

**Returns:**

int: the number of sale prices that were entered incorrectly

**Constraints**

* 1 ≤ n ≤ 10
* 1 ≤ m ≤ n
* ≤ productPrices[i], soldPrice[j] ≤ 100000.00, where 0 ≤ i < n, and 0 ≤ j < m

**Input Format for Custom Testing**

Input from stdin will be processed as follows and passed to the function.The first line contains an integer n the size of the products array.

The next n lines each contain an element products[i].

The next line contains an integer n, the size of the productPrices array.

The next n lines each contain an element productPrices[i].

The next line contains an integer m, the size of the productSold array.

The next m lines each contain an element, productSold[j].

The next line contains an integer, m, the size of the soldPrice array.

The next m lines each contain an element soldPrice[j].

**Sample Case 0**

**Sample Input 0**

**STDIN** Function

4 → products[] size n = 4

rice → products=['rice', 'sugar', 'wheat', 'cheese']

sugar

wheat

cheese

4 → productPrices[] size n = 4

16.89 → productPrices=[16.89, 56.92, 20.89, 345.99]

56.92

20.89

345.99

2 → productSold[] size m = 2

rice → productSold =['rice', 'cheese']

cheese

2 → soldPrice[] size m = 2

18.99 → soldPrice =[18.99, 400.89]

400.89

**Sample Output**

2

**Explanation 0**

Price

Product Actual Expected Error

Rice 18.99 16.89 1

cheese 400.8 9 345.99 1

The sales of rice and cheese were at the wrong prices. So, the number of sale prices that were entered incorrectly is 2

**Sample Case 1**

**Sample Input 1**

STDIN Function

3 → n = 3 .The size of the products array

chocolate → products=[chocolate, cheese, tomato]

cheese

tomato

3 → n = 3 .The size of the productPrices array

15.00 → productPrices=[15.00, 300.90, 23.44]

300.90

23.44

3 → m = 3 .The size of the productSold array

chocolate → productSold=[chocolate, cheese, tomato]

cheese

tomato

3 → m = 3 .The size of the soldPrice array

15.00 → soldPrice =[15, 300.90,10.00]

300.90

10.00

**Sample Output 1**

1

**Explanation 1**

Price

Product Actual Expected Error

chocolate 15.00 15.00

cheese 300.90 300.90

tomato 10.00 23.44 1

Only the tomato sale does not match the price list. So, the number of sale prices that were entered incorrectly is 1.

Hint 1:

Think of a data structure which can help you efficiently store the prices for each of the product.

Ans - Hash Table

Hint 2

If we store the prices of each of the products in the hash table, think about how we can calculate the number of products sold at a wrong selling price.

**Solution**

**Concepts covered:** Basic Programming, Data Structures, Hashing, Loops

The problem tests the candidate's ability to efficiently use the knowledge of the data structures which can store a string, value pair. The candidate is required to come up with a solution to efficiently find the value corresponding to a given string and use conditional operators to check for equality of the value.

**Optimal Solution:**

Instead of iterating over the products for each of the items sold, we can perform some preprocessing to store the prices of each of the products efficiently. A hash table (C++ map or a Python dictionary) can be used here. After preprocessing, we can retrieve the prices of each of the items in O(1).

Time Complexity - O(N + M), here we have assumed that the string size for the name of the products

is small.

def priceCheck(products, productPrices, productSold, soldPrice):

# create a hash map of tuples, key = product, value = correct price

prices = dict((prod, prod\_price) for prod, prod\_price in

zip(products, productPrices))

ans = 0

# iterate the array of sales comparing prices with hash map

for prod, sell\_price in zip(productSold, soldPrice):

if prices[prod] != sell\_price:

ans += 1

return ans

Brute Force Approach: We can iterate over the sold products and for each of the products, we iterate over the list of products to find its actual selling price. If the selling price differs from the sold price, we increment the answer

def priceCheck(products, productPrices, productSold, soldPrice):

ans = 0

for prod, pr in zip(productSold, soldPrice):

idx = products.index(prod)

if pr != productPrices[idx]:

ans += 1

return ans

Time Complexity - O(N x M), where N is the number of products and M is the number of items sold.

CANDIDATE ANSWER

Language used: Java 8

class Result {

/\*

\* 'priceCheck' function.

\*

\* returns an INTEGER.

\* @param products,productPrices, productSold, soldPrice

\*

\*/

public static int priceCheck (List<String> products, List<Float>

productPrices, List<String> productSold, List<Float> soldPrice) {

// Write code here

}