Chinese Stickers

A new collection of Chinese stickers is becoming known in the whole world for its increasing large number of sticker models. Each sticker model is represented with a positive integer, but there is not any predefined limit as everyday new models are becoming available. However, kids usually buy stickers on their town and each town only manufactures a very small number of sticker models.

You are asked to define a **new data structure** for representing decks of stickers. Each deck should contain the number of stickers for each sticker model. Notice that given the features of this Chinese collection, **each deck will have zero stickers for almost every sticker model**. You should consider this for implementing **an efficient data structure**, in which none of the operations should depend on the total number of sticker models (which is undetermined, as we know).

One of the most relevant operations is to calculate the number of shared stickers (i.e. with the same sticker model) between two desks. This operation will allow one to know the number of cards of each deck that can be used by each kid for playing against each other in a fair battle game.

The new data structure should have the following operations:

- Deck(): -> Deck. Builder that creates an empty deck
- addStickers(stickerModel, numStickers): Deck x int x int -> Deck: Modifier function that adds a number of stickers (numStickers) of a given sticker model (stickModel) to the current deck.
- getNumStickers(stickerModel): Deck x int -> int: Observer function that retrieves the number of stickers for a given sticker model.
- numCommonStickers(deck): Deck x Deck -> int: Given the current deck and another deck as parameter, it calculates the number of stickers that are common in both decks. For each sticker model, the number of common stickers will be the lowest value among the two numbers of stickers from the two decks. The total amount of shared stickers will be the sum of all the shared stickers for all the sticker models

In order to represent a deck with a string, we will use the following notation:

[m1 : n1 ; m2 : n2 ; …; mN : nN]

Where

- m1, m2, …, mN are the sticker models for which the number of stickers is greater than zero
- n1, n2, …, nN are the number of stickers for respectively the corresponding sticker models

For example, the following string represents a deck with 3 stickers of model 1000, 2 stickers of model 1777 and 7 stickers of model 2300:

[1000 : 3 ; 1777 : 2 ; 2300 : 7]

For the following two decks, the number of common stickers is 5:

[1000 : 9 ; 1777 : 2 ; 2300 : 7]

[1777 : 4 ; 2300 : 3 ; 2888 : 7 ; 1722832334 : 1]

In this example, the only shared sticker models are 1777 and 2300. In particular, both decks share 2 stickers of 1777 model and 3 stickers of 2300, so the total number of shared stickers is 5.

Input

The first line will indicate the number of cases. Each case will be represented with a line, which contains the string representations of two decks:

<deck1> <deck2>

An example of input case can be the following one:

[1000 : 9; 1777 : 2 ; 2300 :7] [1777 : 4 ; 2300 : 3 ; 2888 : 7 ; 1722832334 : 1]

Output

The output of each case will be printed in a different line. Each output case will indicate the number of shared stickers for each input pair of decks.

Implementation Details

The new data structure should be represented as a **new C++ class** with all the operations indicated in the instructions. It is recommended that you define a **separate function for reading each deck**. This function can be static in the class and create a new deck. Alternatively, you can define this function in the main file, if you prefer so. For comparing if two strings are the same, you can use "!str1.compare(str2)" from "string" library. The "min" function is in "algorithm" library. Implement also a file with the main function for handling the inputs and providing the outputs.

Example of input

```
3
[ 1000 : 9 ; 1777 : 2; 2300 : 7] [ 1777 : 4 ; 2300 : 3 ; 2888 : 7 ; 1722832334 : 1]
[ ] [ 1700 : 11 ; 12345672 : 3 ]
[ 5 : 2 ; 722832334 : 8 ; 722832337 : 3; 1722832339 : 9 ] [ 722832334 : 6 ; 1722832339 : 8 ; 1722832600 : 3 ]
```

Example of output

```
5
0
14
```