1h 18m left

ALL

1. Python: Multiset Implementation

A *multiset* is the same as a set except that an element might occur more than once in a multiset. Implement a multiset data structure in Python. Given a template for the *Multiset* class, implement 4 methods:

- *add(self, val)*: adds *val* to the multiset
- *remove(self, val)*: if *val* is in the multiset, removes *val* from the multiset; otherwise, do nothing
- __contains__(self, val): returns True if *val* is in the multiset; otherwise, it returns False
- __len__(self): returns the number of elements in the multiset

Additional methods are allowed as necessary.

The implementations of the 4 required methods will be tested by a provided code stub on several input files. Each input file contains *several* operations, each of one of the below types. Values returned by *query* and *size* operations are appended to a *result* list, which is printed as the output by the provided code stub.

- add val: calls add(val) on the Multiset instance
- remove val: calls remove(val) on the Multiset instance
- query val: appends the result of expression *val* in *m,* where *m* is an instance of Multiset, and appends the value of that expression to the *result* list
- size: calls len(m), where *m* is an instance of Multiset, and appends the returned value to the *result* list

Complete the class Multiset in the editor below with the 4 methods given above (add, remove, __contains__, and __len__).

Constraints

- $1 \leq$ number of operations in one test file $\leq 10^5$
- if *val* is a parameter of operation, then *val* is an integer and $1 \leq val \leq 10^9$

▶ Input Format Format for Custom Testing