4 Set with range sums

Problem Introduction

In this problem, your goal is to implement a data structure to store a set of integers and quickly compute range sums.

Problem Description

Task. Implement a data structure that stores a set S of integers with the following allowed operations:

- **add**(i) — add integer i into the set S (if it was there already, the set doesn't change).
- **del**(i) — remove integer i from the set S (if there was no such element, nothing happens).
- **find**(i) — check whether i is in the set S or not.
- **sum**(l, r) — output the sum of all elements v in S such that $l \leq v \leq r$.

Input Format. Initially the set S is empty. The first line contains n — the number of operations. The next n lines contain operations. Each operation is one of the following:

- "+ i" — which means **add** some integer (not i, see below) to S,
- "- i" — which means **del** some integer (not i, see below)from S,
- "? i" — which means **find** some integer (not i, see below)in S,
- "s l r" — which means compute the **sum** of all elements of S within some range of values (not from l to r, see below).

However, to make sure that your solution can work in an online fashion, each request will actually depend on the result of the last **sum** request. Denote $M = 1\ 000\ 000\ 001$. At any moment, let x be the result of the last **sum** operation, or just 0 if there were no **sum** operations before. Then

- "+ i" means **add**$((i + x) \bmod M)$,
- "- i" means **del**$((i + x) \bmod M)$,
- "? i" means **find**$((i + x) \bmod M)$,
- "s l r" means **sum**$((l + x) \bmod M, (r + x) \bmod M)$.

Constraints. $1 \leq n \leq 100\ 000$; $0 \leq i \leq 10^9$.

Output Format. For each find request, just output "Found" or "Not found" (without quotes; note that the first letter is capital) depending on whether $(i + x) \bmod M$ is in S or not. For each **sum** query, output the sum of all the values v in S such that $((l+x) \bmod M) \leq v \leq ((r+x) \bmod M)$ (it is guaranteed that in all the tests $((l + x) \bmod M) \leq ((r + x) \bmod M)$), where x is the result of the last **sum** operation or 0 if there was no previous **sum** operation.

Time Limits.

language	C	C++	Java	Python	C#	Haskell	JavaScript	Ruby	Scala
time (sec)	1	1	4	120	1.5	2	120	120	4

Memory Limit. 512MB.

Sample 1.

Input:

```
15
? 1
+ 1
? 1
+ 2
s 1 2
+ 1000000000
? 1000000000
- 1000000000
? 1000000000
s 999999999 1000000000
- 2
? 2
- 0
+ 9
s 0 9
```

Output:

```
Not found
Found
3
Found
Not found
1
Not found
10
```

Explanation:
For the first 5 queries, $x = 0$. For the next 5 queries, $x = 3$. For the next 5 queries, $x = 1$. The actual list of operations is:
find(1)
add(1)
find(1)
add(2)
sum$(1, 2) \to 3$
add(2)
find$(2) \to$ Found
del(2)
find$(2) \to$ Not found
sum$(1, 2) \to 1$
del(3)
find$(3) \to$ Not found
del(1)
add(10)
sum$(1, 10) \to 10$

Adding the same element twice doesn't change the set. Attempts to remove an element which is not in the set are ignored.

Sample 2.

Input:

```
5
? 0
+ 0
? 0
- 0
? 0
```

Output:

```
Not found
Found
Not found
```

First, 0 is not in the set. Then it is added to the set. Then it is removed from the set.

Sample 3.

Input:

```
5
+ 491572259
? 491572259
? 899375874
s 310971296 877523306
+ 352411209
```

Output:

```
Found
Not found
491572259
```

Explanation:
First, 491572259 is added to the set, then it is found there. Number 899375874 is not in the set. The only number in the set is now 491572259, and it is in the range between 310971296 and 877523306, so the sum of all numbers in this range is equal to 491572259.

Starter Files

The starter solutions in C++, Java and Python3 read the input, write the output, fully implement splay tree and show how to use its methods to solve this problem, but don't solve the whole problem. You need to finish the implementation. If you use other languages, you need to implement a solution from scratch.

Note that we strongly encourage you to use stress testing, max tests, testing for min and max values of each parameter according to the restrictions section and other testing techniques and advanced advice from this reading. If you're stuck for a long time, you can read ths forum thread to find out what other learners struggled with, how did they overcome their troubles and what tests did they come up with. If you're still stuck, you can read the hints in the next What to Do section mentioning some of the common problems and how to test for them, resolve some of them. Finally, if none of this worked, we included some of the trickier test cases in the starter files for this problem, so you can use them to debug your program if it fails on one of those tests in the grader. However, you will learn more if you pass this problem without looking at those test cases in the starter files.

What to Do

Use splay tree to efficiently store the set, add, delete and find elements. For each node in the tree, store additionally the sum of all the elements in the subtree of this node. Don't forget to update this sum each time the tree changes. Use split operation to cut ranges from the tree. To get the sum of all the needed elements after split, just look at the sum stored in the root of the splitted tree. Don't forget to merge the trees back after the **sum** operation.

Some hints based on the problems some learners encountered with their solutions:

- Use the sum attribute to keep updated the sum in the subtree, don't compute the sum from scratch each time, otherwise it will work too slow.
- Don't forget to do splay after each operation with a splay tree.
- Don't forget to splay the node which was accessed last during the find operation.
- Don't forget to update the root variable after each operation with the tree, because splay operation changes root, but it doesn't change where your root variable is pointing in some of the starters.
- Don't forget to merge back after splitting the tree.
- When you detach a node from its parent, don't forget to detach pointers from both ends.
- Don't forget to update all the pointers correctly when merging the trees together.
- Test sum operation when there are no elements within the range.
- Test sum operation when all the elements are within the range.
- Beware of integer overflow.
- Don't forget to check for null when erasing.
- Test: Try adding nodes in the tree in such an order that the tree becomes very unbalanced. Play with this visualization to find out how to do it. Create a very big unbalanced tree. Then try searching for an element that is not in the tree many times.
- Test: add some elements and then remove all the elements from the tree.

Need Help?

Ask a question or see the questions asked by other learners at this forum thread.