Balanced Search Tree

In this problem, you are required to implement a data structure to support the following operations.

The files attached to this problem are shown as follows. We provide some tests to help you check your code.

```
.
└── attachments/
    ├── tests/
    |   ├── insert.cpp
    |   ├── erase.cpp
    |   ├── querykth.cpp
    |   ├── queryksum.cpp
    |   └── rebuild.cpp
    ├── scapegoat.pdf
    └── bst.hpp
```

NOTE: You can not use STL data structures, such as `std::unordered_map`, `std::unordered_set`, `std::map` and `std::set`, in this problem. **All of your data structure should be designed by yourself.**

Requirements:

Description:

You've got a set, consisting of integers. Assume before each operation, the set contains m integers, which are $s_1 \le s_2 \le s_3 \le \ldots \le s_m$. You are allowed to perform five operations on this set:

1. Given x, insert x into the set
2. Given x, erase x from the set
3. Given x, query whether x is in the set
4. Given k, query the k-th element of set i.e. s_k.
5. Given k, you need to find a position t such that $\left(k\sum_{i=1}^{t} s_i\right) - \sum_{i=t+1}^{m} s_i \ge 0$, meanwhile minimize $\left(k\sum_{i=1}^{t} s_i\right) - \sum_{i=t+1}^{m} s_i$. You should return the minimum ans. In other words:

$$ans = \min_{t\in[1,m]} \sum_{i=1}^{t} k \cdot s_i - \sum_{i=t+1}^{m} s_i \qquad \text{s.t.} \sum_{i=1}^{t} k \cdot s_i - \sum_{i=t+1}^{m} s_i \ge 0$$

Test inference introduction

Write your code in `bst.hpp`.

Our checker would first run the function `void init(std::size_t n)`. Here, `std::size_t n` is the number of operations we would do in this test case. In this function, you can do anything to initialize your data structure. We would guarantee that this function would be called first and only once in each test case.

Next, our checker would call `long long SetOperation(std::size_t opt, long long x)` exactly n times sequentially. You need to return answer values through this function call.
After each call, we would check your answer. Only if your i-th answer of `SetOperation` is correct, could you receive the $i+1$-th function call. This means you cannot implement any off-line algorithm method in this

problem.

In `SetOperation`, you should do operations on your data structure according to (opt, x). Assume in the i-th call, there are $m (0 \le m \le i-1)$ elements in the set, which are $s_1 \le s_2 \le \ldots \le s_m$.

There are only 5 kinds of operations:

- if $opt = 1$, you need to insert **exactly one** $x (1 \le x \le 10^{11})$ into your data structure and return 0.
- if $opt = 2$, you need to erase **exactly one** x from your data structure and return 0. It is guaranteed that there exists $1 \le j \le m$ such that $s_j = x$.
- if $opt = 3$, you need to answer whether $x (1 \le x \le 10^{11})$ is in the data structure. If x is in the set, return 1. Otherwise, return 0.
- if $opt = 4$, it is guaranteed that $1 \le x \le m$, and you should return s_x.
- if $opt = 5$, it is guaranteed that $1 \le x \le 100$ and $m \ge 1$, and you need to return the answer of query, where ans is

$$ans = \min_{t \in [1,m]} \sum_{i=1}^{t} x \cdot s_i - \sum_{i=t+1}^{m} s_i \qquad \text{s.t.} \sum_{i=1}^{t} x \cdot s_i - \sum_{i=t+1}^{m} s_i \ge 0$$

If you successfully pass all the set operations, our checker would call `void clear()`, where you **MUST** deconstruct your data structure and release your resources.

Data constraints

In all test cases, it is guaranteed that $1 \le n \le 400000, 1 \le opt \le 5, 1 \le x \le 10^{11}$.

Please pay attention to the special time limit and memory limit.

Time complexity

If you implement your algorithm in $\Theta(n^2)$ time, you will receive about 40 pts.

If you implement your algorithm in $\Theta(n \log^2 n)$ or $\Theta(n \log x)$time, you will receive about 70 pts.

If you implement your algorithm in $O(n \log n)$ time, you will receive 100 pts.

Space complexity

If you implement your algorithm in $\Theta(n \log n)$ space, you will receive at most 70 pts.

To fully pass this problem, you should implement your algorithm in $O(n)$ space.

Hint

You can use any self-balancing binary search tree to solve this problem, e.g. Scapegoat tree, AVL tree, Red-black tree, Splay tree, Treap and B-tree.

We recommend you implement Scapegoat tree. Scapegoat tree is a non-rotating binary tree. Its balancing is achieved by completely rebuild the "unbalanced" subtree into a complete binary tree. This makes Scapegoat tree easier to implement than AVL tree and Red-black tree.

Please see more information in scapegoat.pdf.

提交与评分

在 OJ 上提交 `bst.hpp` 内的全部内容。

本题的评分由 OJ 分数（60%）和线下 check （40%）两部分构成。线下 check 会在此次作业结束时间之后进行。

注：线下 check 也带有检查学术诚信的含义，当然这不是唯一的手段。如果被认定为抄袭， OJ 的分数也会作废，并且会有惩罚。**特别强调，抄袭来自 generative AI 的代码和抄袭网上的代码是同等处理的，我们建议您在写作业时关闭一切**
generative AI 工具。