Assignment 3

Due Date: Monday, April 10th 2017, 11:59pm

Objectives

In this assignment, you will gain a deeper familiarity with balanced binary trees. Your solution should use a balanced binary tree (such as AVL Tree) but researching another balanced binary tree and implementing it is allowed.

Beware: The binary search tree shown in class had a bug in delete. Find a way to fix this or implement a better delete. The tree in this problem is also not a search tree.

Deliverables

Submit two files.

The file named "`fancy.h`" is your header file. Place all function declarations in this header with comments on how you designed the header. You may use the header I provide or design your own. In either case, include a header.

The file named is "`fancy.c`" is your source file. Implement all functions defined in the header file. Include a main function in this file.

All input should be read from a file named "`fancy.txt`". The input specification will define precisely what is allowed for input to your program. You need not guard against cases not allowed in the input.

Scoring

I will run your program multiple times on multiple test files. Your grade will be awarded based on the number of tests you pass with some points allocated to fulfilling the special requirements for sorting. Be sure to test your program on more than just the sample cases provided.

To guard against infinite loops and inefficient implementations, your program will be run for x seconds per test case. Upon grading, you will receive a "`results.txt`" file detailing which tests where passed and which failed. Here is a table of possible results:

Error Code	Meaning
AC	Accepted! :D All is wonderful in the world and your code worked correctly.
WA	Wrong Answer: The code executed but produced faulty output.
RTE	Runtime Error: The code crashed spewing memory and guts everywhere.
TLE	Time Limit Exceeded: I killed the program after x seconds. I feel betrayed.
CTE	Compile Time Error: Your code did not compile. ☹

Fancy Array

`filename: fancy`
`timelimit: 3 seconds`

In this problem, you will implement a fancy version of an array using a binary search tree. We want a data structure that can do each of the following:

- Insert some integer into position i (sliding all values at j where $i \leq j$ to the right)
- Delete some integer from position i (sliding all values at j where $i < j$ to the left)
- Obtain the value at index i.

Each of these operations should work in $O(\log n)$ worst case runtime. (This is why it is a fancy array.)

Input "fancy.txt"

The first line of input contains a single integer n ($1 \leq n \leq 10^6$) representing the number of operations to process.

This is followed by n lines, with one operation per line.

- valueat i (print the value at position i in the array, print "error" if no such value exists)
- insert i v (insert the value v into position i in the array)
- delete i (delete the value at position i from the array)

If the current array size is m, you may assume that i in the valueat and delete command will be an integer in the range $[0, m-1]$. You may assume that position i in an insert will be in the range $[0, m]$. The value of v, inserted into the array, will be an integer in the range $[-10^9, 10^9]$.

Output (stdout)

Output one line for each $valueat$ command, the value at that position in the array when the command was issued.

Sample 1

Input	8 insert 0 1 insert 0 2 insert 0 3 insert 0 4 delete 2 valueat 0 valueat 1 valueat 2
Output	4 3 1

Sample 2

Input	22 insert 0 1 insert 1 2 insert 1 4 insert 3 3 insert 2 5 valueat 0 valueat 1 valueat 2 valueat 3 valueat 4 delete 2 valueat 2 valueat 3 delete 0 valueat 0 delete 0 valueat 0 delete 0 valueat 0 delete 0 insert 0 100 valueat 0
Output	1 4 5 2 3 2 3 4 2 3 100

Tips

To balance AVL Trees you can store and maintain height information in your tree. To implement the indexing, you can store the size of each subtree. Here the size is the number of nodes in that subtree. Think about how this helps you find the i^{th} node.