Programming Assignment 3
Due: November 17, 2017

0 Instructions

Submit your work through Canvas. You should submit a tar file containing all source files and a README for running your project.

More precisely, submit on Canvas a tar file named lastname.tar (where lastname is your last name) that contains:

- All source files. You can choose any language that builds and runs on ix-dev.

- A file named README that contains your name and the exact commands for building and running your project on ix-dev. If the commands you provide don't work on ix-dev, then your project can't be graded and you won't receive credit for the assignment.

Here is an example of what to submit:

hampton.tar
  problem1-1.py
  problem1-2.py
  ...
  README

```
README
  Andrew Hampton

  Problem 1.1: python problem1-1.py input.txt
  Problem 1.2: python problem1-2.py input.txt
  ...
```

Note that Canvas might change the name of the file that you submit to something like lastname-N.tar. This is totally fine!

The grading for this assignment will be roughly as follows:

Task	**Points**
Problem 1	50
Problem 2 (extra)	5
Problem 3 (extra)	5
TOTAL	50

1 Implementation

Problem 1. Red-Black Tree
For this problem, you will implement a red-black tree with integer keys. Do not use any builtin tree structures that your language might have. You must implement your own binary search tree class that satisfies the red-black properties as described in Chapter 13 of the textbook.

Your data structure should implement the methods:

`insert(X)`: Inserts a node into the tree having key X. Runtime: $O(\log n)$

`remove(X)`: If the key X is present, removes a node having key X from the tree. Runtime: $O(\log n)$

`search(X)`: Returns a boolean indicating whether the key X is present. Runtime: $O(\log n)$

`maximum()`: Returns an integer, the largest key in the tree. Does not alter the tree. Runtime: $O(\log n)$

`minimum()`: Returns an integer, the smallest key in the tree. Does not alter the tree. Runtime: $O(\log n)$

Optional: Implement the following method. Note that the extra credit problems can be submitted only if you complete this optional step.

`select(k)`: Returns the kth smallest key, indexed from zero. That is, select(0) returns the smallest key in the tree, select(1) returns the next smallest, etc. Runtime: $O(\log n)$

Hint: in order to achieve the desired runtime for select, try storing at each node the size of the subtree rooted at that node and maintain this property as an invariant.

Write a driver program that takes a single command-line argument, which will be a filename. The input file will contain instructions for tree operations. The first line of the input file will be an integer $0 \leq N \leq 10^6$ giving the number of instructions. Following will be N lines, each containing an instruction. The possible instructions are:

`insert X`, where $-10^5 \leq X \leq 10^5$ is an integer: insert a node with key X into the tree. There is no output.

`remove X`, where $-10^5 \leq X \leq 10^5$ is an integer: remove a node with key X from the tree. If such a node exists, there is no output. If no such node exists, output *TreeError*.

`search X`, where $-10^5 \leq X \leq 10^5$ is an integer: output *Found* if a node exists with key X. If no such node exists, output *NotFound*.

`max`: output the maximum key in the tree. If the tree is empty, output *Empty*.

`min`: output the minimum key in the tree. If the tree is empty, output *Empty*.

`print`: print the keys of the tree according to an inorder traversal, separated by a single space. If the tree is empty, output *Empty*.

For the print action, you should implement a `print_tree` function that takes a tree as an argument and accomplishes the print action. The runtime should be $O(n)$.

Example input file:

```
18
print
remove 2
max
search 5
insert 1
insert 2
search 1
search 2
insert 3
print
insert 10
insert 5
print
search 5
remove 2
print
max
min
```

Example output:

```
Empty
TreeError
Empty
NotFound
Found
Found
1 2 3
1 2 3 5 10
Found
1 3 5 10
10
1
```

2 Applications

Problem 2. Extra Credit: Rolling Median 2
This will build on the rolling median problem from the previous homework. Read the description of that problem again to recall the details. Previously, we only added integers to the set and re-computed the median. What if we also wanted to remove integers? The heap solution can't do it without a serious runtime penalty!

In this problem, you will need to solve the same rolling median problem as the previous assignment, except that some of the instructions in the input file will require removing an integer from the set under consideration.

Note: In order to receive credit for this problem, you need to implement a method `median` on your red-black tree data structure that returns the median of the keys in $O(\log n)$ time. Use this method in your solution.

Your program should take a single command-line argument, which will be a filename. The input file will contain integers, one per line. The first line of the input file will be an integer $2 \leq N \leq 10^5$ giving the number of integers in the list L. Following will be N lines, each containing an instruction of the form *add* x or *remove* x, where $0 \leq x \leq 10^6$ is an integer. If the instruction is *add* x, then add x to the set of integers under consideration. If the instruction is *remove* x, then remove x from the set of integers under consideration.

For each instruction, you should output the median of the set of integers under consideration. A median should be printed to one decimal place if and only if it is not an integer. See the sample output below.

RUNTIME: Your solution should have runtime complexity $O(n \log n)$.

Hint: Use a single red-black tree and use the `select` method to efficiently implement the `median` method.

Example input file:

```
8
add 1
add 8
add 4
add 3
add 2
remove 3
remove 1
add 10
```

Example output:

```
1
4.5
4
3.5
3
3
4
6
```

Problem 3. Extra Credit: Rolling Median 3
One more rolling median problem!

The input and output are exactly the same as the previous problem, but in this problem you need to implement a different solution.

You implemented `minimum` and `maximum` methods on your red-black tree having runtime $O(\log n)$. This means that the red-black tree could be a drop-in replacement for the minheap and maxheap in the suggested solution for the rolling median problem from Programming Assignment 2.

Try it out! Solve the extended rolling median problem using two red-black trees, one to keep track of the smaller half of the data and one to keep track of the larger half of the data. Don't use the `median` method that you implemented for the previous problem.

RUNTIME: Your solution should have runtime complexity $O(n \log n)$.

You now have three solutions to the rolling median problem! Time them to see which is the fastest, using either the UNIX *time* command or a builtin timing library (see the lab notes for a Python timing example). Write a short comment (just a few sentences!) in your source code explaining the test cases you chose to time and the meaning of your results.

Example input file:

```
8
add 1
add 8
add 4
add 3
add 2
remove 3
remove 1
add 10
```

Example output:

```
1
4.5
4
3.5
3
3
4
6
```