

CS 201 Data Structures Library Phase 2 Due 10/18

Phase 2 of the CS201 programming project, we will be built around a balanced binary search tree. In particular, you should implement red-black trees as described in the CLRS textbook for the class `RBTree`.

The public methods of your class should include the following (keytype and valuetype indicate the types from the template):

Function	Description	Runtime
<code>RBTree();</code>	Default Constructor. The tree should be empty	$O(1)$
<code>RBTree(keytype k[], valuetype V[], int s);</code>	For this constructor the tree should be built using the arrays K and V containing s items of keytype and valuetype.	$O(s \lg s)$
<code>~RBTree();</code>	Destructor for the class.	$O(n)$
<code>valuetype * search(keytype k);</code>	Traditional search. Should return a pointer to the valuetype stored with the key. If the key is not stored in the tree then the function should return NULL.	$O(\lg n)$
<code>void insert(keytype k, valuetype v);</code>	Inserts the node with key k and value v into the tree.	$O(\lg n)$
<code>int remove(keytype k);</code>	Removes the node with key k and returns 1. If key k is not found then remove should return 0. If the node with key k is not a leaf then replace k by its predecessor.	$O(\lg n)$
<code>int rank(keytype k);</code>	Returns the rank of the key k in the tree. Returns 0 if the key k is not found. The smallest item in the tree is rank 1.	$O(\lg n)$
<code>keytype select(int pos);</code>	Order Statistics. Returns the key of the node at position pos in the tree. Calling with $\text{pos} = 1$ should return the smallest key in the tree, $\text{pos} = n$ should return the largest.	$O(\lg n)$
<code>keytype *successor(keytype k)</code>	Returns a pointer to the key after k in the tree. Should return NULL if no successor exists.	$O(\lg n)$
<code>keytype *predecessor(keytype k)</code>	Returns a pointer to the key before k in the tree. Should return NULL if no predecessor exists.	$O(\lg n)$
<code>int size();</code>	returns the number of nodes in the tree.	$O(1)$
<code>void preorder();</code>	Prints the keys of the tree in a preorder traversal. The list of keys are separated by a single space and terminated with a newline.	$O(n)$
<code>void inorder();</code>	Prints the keys of the tree in an inorder traversal. The list of keys are separated by a single space and terminated with a newline.	$O(n)$
<code>void postorder();</code>	Prints the keys of the tree in a postorder traversal. The list of keys are separated by a single space and terminated with a newline.	$O(n)$

void printk(int k);	Prints the smallest k keys in the tree. The list of keys are separated by a single space and terminated with a newline.	$O(k + \lg n)$
---------------------	-------------------------------------------------------------------------------------------------------------------------	----------------

Your class should include proper memory management, including a destructor, a copy constructor, and a copy assignment operator.

For submission, all the class code should be in a file named `RBTree.cpp`. Create a makefile for the project that compiles the file **Phase2Main.cpp** and creates an executable named **Phase2**. A sample makefile is available on Blackboard. Place both `RBTree.cpp` and makefile into a zip file and upload the file to Blackboard.

- ☐ Create your `RBTree` class
- ☐ Modify the makefile to work for your code (changing compiler flags is all that is necessary)
- ☐ Test your `RBTree` class with the sample main provided on the cs-intro server
- ☐ Make sure your executable is named `Phase2`
- ☐ Develop additional test cases with different types, and larger trees
- ☐ Create the zip file with `RBTree.cpp` and makefile
- ☐ Upload your zip file to Blackboard

No late submissions will be accepted.