CS 201 Data Structures Library Phase 2 Due 3/23

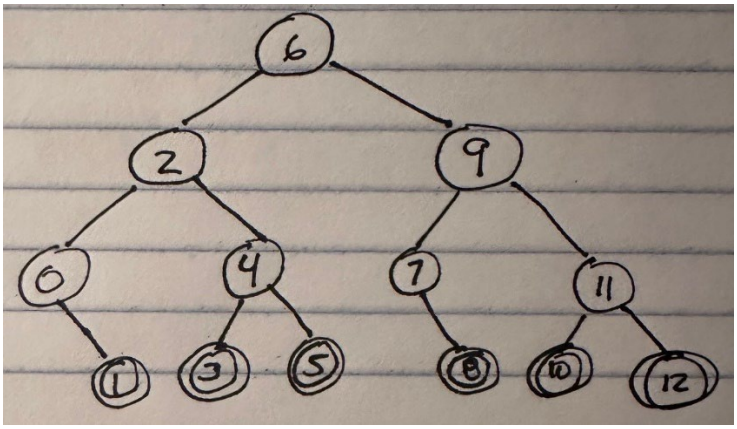
Phase 2 of the CS201 programming project, we will be built around a balanced binary search tree. In particular, you should implement red-black trees as described in the CLRS textbook for the class RBTree.

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

Function	Description	Runtime
RBTree();	Default Constructor. The tree should be empty	O(1)
RBTree(keytype k[], valuetype V[], int s);	For this constructor the tree should be built using the arrays K and V containing s items of keytype and valuetype. If the keytype array is order then the tree should be built in O(s) time as described later. If the items are not in order, the tree should be built using repeated insertion.	O(s) if in order O(s lg s) else
~RBTree();	Destructor for the class.	O(n)
valuetype * search(keytype k);	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)
void insert(keytype k, valuetype v);	Inserts the node with key k and value v into the tree.	O(lg n)
int remove(keytype k);	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)
int rank(keytype k);	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)
keytype select(int pos);	Order Statistics. Returns the key of the node at position pos in the tree. Calling with pos = 1 should return the smallest key in the tree, pos = n should return the largest.	O(lg n)
keytype *successor(keytype k)	Returns a pointer to the key after k in the tree. Should return NULL if no successor exists.	O(lg n)
keytype *predecessor(keytype k)	Returns a pointer to the key before k in the tree. Should return NULL if no predecessor exists.	O(lg n)
int size();	returns the number of nodes in the tree.	O(1)
void preorder();	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)

void inorder();	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)
void postorder();	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)

When the constructor is given an array of keys in order then the tree should be built as follows. Suppose that the array has 2k – 1 elements. In this case the tree will be a complete binary tree with all elements colored black. For instance, if the array has 7 items in it with indices 0…6 then item 3 will be the root with item 1 as the left child and item 5 as the right child. Now we generalize this process by choosing the item that splits the items in the subtree as close to in half as possible, but choose the lower index when there is not an exact median. In this case the tree will be a complete binary tree except at the lowest level. Color the nodes at the lowest level red and the rest black. The tree below is an example containing 13 items with indices 0…12:



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.