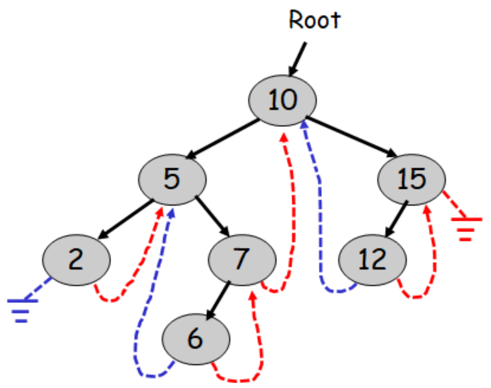
Due on: December 30, 2020

Data Structures

Lab5: Threaded BST

You are asked to implement a Threaded Binary Search Tree. A threaded BST is a BST where a the normally NULL left/right child pointers of a BST node point to the in-order predecessor/in-order successor node in the BST respectively. This allows us to walk over the threaded BST in forward or backward direction iteratively in O(n) time, where "n" is the number of keys in the tree. Here is an example threaded BST containing the keys 10, 5, 15, 2, 7, 12 and 6:



Pay special attention to node that have 0 or 1 child. In a BST, the pointers pointing to a non-existent child would be set to NULL. But in the threaded BST, the right child pointer that would normally be NULL points to the in-order successor node. Similarly, a left child pointer that would normally be NULL points to the in-order predecessor node. If the in-order predecessor/successor does not exist, then and only then is the corresponding pointer set to NULL.

In the given example, node 2 is a leaf node with 0 children. Since it does not have an in-order predecessor, its left pointer is set to NULL. Its right pointer points to node 5, which is 2's in-order successor.

Node 6 is also a leaf node with 0 children. Its left pointer points to node 5, which is 6's in-order predecessor; and its right pointer points to 7, which is 6's in-order successor.

Node 7 has a left child, but not right child. Therefore, node 7's right pointer points to 10, which is 7's in-order successor.

If a node has any children, then its child pointers point to its respective child as in a regular BST. For example, 10 has a left child 5 and a right child 15, and its left and right pointers point to these children respectively.

In this lab, you are asked to change the BST code we have given you so that it becomes a threaded BST as described above. Here is the decleration for BSTNode structure:

```
#define CHILD           1
#define THREAD          2

struct BSTNode {
        int key;
        BSTNode* left, * right;
        char leftLinkType, rightLinkType; // The link type for the left/right pointers (CHILD or THREAD)

        // When the node is first created, it is a leaf node. So, both links must be THREAD
        BSTNode(int key) { this->key = key; left = right = NULL; leftLinkType = rightLinkType = THREAD; }
};
```

Notice that we have to distinguish between a real child pointer and a THREAD pointer. When left/right points to a real-child, then leftLinkType/rightLinkType must be set to CHILD. If left points to the in-order predecessor, then leftLinkType must be set to THREAD. Similarly, when right points to the in-order successor, then rightLinkType must be set to THREAD.

The interface for your ThreadedBST class is the following:

```
class ThreadedBST {
private:
        BSTNode* root;

        void eraseTreeNodes(BSTNode* node);
public:
        ThreadedBST() { root = NULL; }
        ~ThreadedBST() { eraseTreeNodes(root); root = NULL; }

        void add(int key);
        void remove(int key);

        BSTNode* getRoot() { return root; }
        BSTNode *find(int key);
        BSTNode* min();
        BSTNode* max();
        BSTNode* previous(BSTNode* node);
        BSTNode* next(BSTNode* node);
};
```

“add” and “remove” methods are used to add/remove a key to/from the Threaded BST. After an add/remove, all links and threads must be adjusted properly.

“find” returns a pointer to the node that contains the given key if it exists. Otherwise it returns NULL.

“min” returns a pointer to the node that contains the minimum key in the tree if it exists. Otherwise it returns NULL.

“max” returns a pointer to the node that contains the maximum key in the tree if it exists. Otherwise it returns NULL.

“previous” takes a valid pointer to a tree node and returns a pointer to the node that contains the in-order predecessor of this node. For example, if “node” points to 12, then its in-order predecessor is 10, which can be found by following the left thread. If “node” points to 10 however, its in-order predecessor is 7, which is the maximum node on its left sub-tree. Thus, you have to know whether the “left” pointer of a node is a real child link, or a thread link to implement this method correctly. That’s why BSTNode contains the link type.

“next” takes a valid pointer to a tree node and returns a pointer to the node that contains the in-order successor of this node. For example, if “node” points to 7, then its in-order successor is 10, which can be found by following the right thread. If “node” points to 10 however, its in-order successor is 12, which is the minimum node on its right sub-tree. Thus, you have to know whether the “right” pointer of a node is a real child link, or a thread link to implement this method correctly. That’s why BSTNode contains the link type.

main.cpp contains two tests to test your ThreadedBST. You are encouraged to implement your own test code. We will add other tests when grading your submissions.