Report

As mentioned in the overview of this phase from the project, we tried to construct an AVL tree to act as a shelf for the shreds. The index of the shreds is related to the shred’s names.

For us to maintain the AVL tree, we tried to handle the cases that may threatens the AVL tree’s balance.

* For us to handle the case of inserting a node to the left of the left node of the root(descending), a right rotation of the left subtree of the root is applied to maintain the balance. Also, the height of the tree is updated.
* For us to handle the case of inserting a node to the right of the right node of the root(ascending), left rotation of the right subtree of the root is applied to maintain the balance. Also, the height of the tree is updated.
* To handle the case of inserting a node to the right of the left node of the root, a left rotation is applied to the right subtree of the left child of the root. Here the case becomes descending then it is handled as previously-mentioned and the height of the tree is updated.
* To handle the case of inserting a node to the left of the right node of the root, a right rotation is applied to the left subtree of the right child of the root. Here the case becomes descending then it is handled as previously-mentioned and the height of the tree is updated
* To find an item, a find method is constructed.
* First, the root is checked whether it’s NULL or not. If it isn’t NULL, root is assigned to the parent node

if ( root != NULL) {

NODE \* parent = root;

NODE \* next = (NODE \*) parent->compareAndAdvance(item);

while ( next != parent) {

if ( next == NULL ) break;

else parent = next;

next = (NODE \*) parent->compareAndAdvance(item);

}

if ( next == parent ) return next;

}

return NULL;