Creating the 'tsq' Package

Andrew Faust, Chi Kei Loi, Mano Wared, Tingting Zhu

March 21, 2017

1 Description

The purpose of creating the tsq package was to implement the Stack, Queue, and Binary Tree data structures in R. However, creating these data structures proved rather difficult, due to Rs lack of pointers. Fortunately, this obstacle was avoided thanks to Rs vector operations. The data structures in this package were all implemented using vectors and R6 reference classes.

2 Stack and Queue

When creating the Stack and Queue data structures, a lack of pointers is not a huge issue. Both data structures can be thought of as arrays (or vectors), with the last element being removed for Stacks, and the first element being removed for Queues when popping. Therefore, that is exactly how these data structures are implemented in this package. When being pushed to a Stack or a Queue, the element is simply appended to the end of the vector. When popping, Stack removes and returns the last element of the vector, while Queue removes and returns the first. Space and time were not much of an issue, as the elements to be popped are always at the front or back of the vector, and R's vector operations allow these elements to be easily removed.

3 Binary Tree

Creating a Binary Tree without pointers proved much more of a challenge than implementing Stack and Queue. Typically, Binary Trees are implemented by using nodes that point to children nodes containing values lesser

or greater than their own. Using Rs matrix operations, however, allowed for a similar concept to be implemented without the use of pointers.

The Binary Tree itself is represented using a single matrix. Each row of the matrix represents a node in the tree, and the matrix has three columns: one for the element within the node, and two for the row indices of the lesser and greater children nodes. The tree initially consists of one row with all null elements, with rows being appended whenever an element is pushed onto the tree. When popping, the parent looks ahead to see if its left child had a left child itself. If not, then the parent updates its left pointer to bypass the left child, while leaving the child in the matrix.

While implementing the pop function, space and time efficiency had to be considered. Would it be better to replace rows that have been popped, or to simply leave them be as their parents would be updated to avoid them? The former would prove to be much more space efficient, however time efficiency would take a hit as the entire matrix would possibly have to be searched before a popped row was found, if any. The latter, however, would be a problem for space, as popped nodes are never truly removed from the tree. After careful consideration, we chose the latter option, and thus our Binary Tree data structure is time efficient, but not very efficient in terms of space.

4 Why R6 Classes?

When implementing these three data structures, we realized that the specification of the pop function conflicts with the convention of the S3 class, and thus ran into a problem when using them. Since R is a functional language, functions have no side effects, meaning that changing the data structures object in the pop function only changes the copy that was passed into the function, and not the object itself. Thus, the pop function needed to return the modified object so that the actual object can be reassigned. By doing so, we cannot return the popped value, which contradicts the implementation of traditional pop method. As a result, we decided to use R6, a powerful library that contains implementation of classes with reference semantics. This made implementing the traditional pop function much easier as we had direct access to the data structure, and could modify it and output the popped value in the same function.

5 Conclusion

These classes are examples of the downsides of R, and how they can be overcome. Using R's vector operations, we were able to mimic the use of pointers to create Stack, Queue, and Binary Tree classes. We were also able to overcome the limitations of the traditional S3 class by using the R6 reference class instead.