Justin Hughes

Lab 60 – 61 Comparison Paper

We will be analyzing the difference between implementations of binary trees using two structures. One binary tree will use Java’s built in Array List data structure. The second binary tree will use a Linked List format data structure. Each one comes with their advantages and disadvantages.

For array-type binary trees, each part of the binary tree corresponds with a specific position within the array. The root sits at index 0 and every following left child corresponds with position parentNode \* 2 + 1 while every following right child corresponds with position parentNode \* 2 + 2. Conversely, child nodes can find their parent node using the equation (index -1) / 2 and then rounding down. Thus, each part of a tree corresponds with a specific index in the array meaning there can be many null / empty values within the array between actual values.

This structure leads to three core benefits. First, in complete trees, the array is space efficient because each position in the array will be filled. Second, accessing elements within the tree is O(1) time because we are simply accessing a specific index within an array. Third, we can traverse the tree with a relatively easy method using the above calculations.

The detriments for an array-based tree really come to the forefront when we talk about trees with are not “complete”. As stated prior, non-complete trees can lead to many empty values within the array. This could lead to quite a bit of wasted space. Also, whenever we reach the maximum capacity of this array, we will need to do an O(n) operation when we must re-size the array’s size.

For Linked List-type binary trees, each value within the tree is a node containing a specific element. The nodes will hold pointers to their respective parent, left child, and right child. In comparison to the array-based tree, this data structure allows for better space efficiency in non-complete trees. Each of the nodes takes up their own space in the memory, and there is no need to resize the underlying data format because we simply create new nodes and pointers to said nodes as we expand the tree.

While the linked list-based tree beats out the array-based tree in terms of space efficiency, it loses out in terms of time to find elements. To find an element within the tree, we must traverse the tree to get to specific positions. If we don’t know where a specific node could be, this could lead to an O(n) time whenever we need to find a node. Another detriment of the tree would be the pointer management as we add and remove nodes from the tree itself. While this is largely a code it once kind of aspect, this can be difficult to grasp for newer programmers and hard to implement for them.

Altogether, the two implementation types have their trade-offs. Arrays tend to win out in terms of speed and simplicity while linked lists tend to be more space efficient especially in cases when trees aren’t complete such as binary search trees.