Module 5: Critical Thinking

Heaps

Nolan Byrnes

CSC506 – Design and Analysis of Algorithms

Colorado State University – Global Campus

Professor Isaac Gang

October 14th, 2022

Module 5: Critical Thinking

Lysecky defines a max heap as ”a binary tree that maintains the simple property that a node's key is greater than or equal to the node's childrens' keys” (Lysecky, 2019a, para. 2). By having each node greater than or equal to its children, in a max-heap, you will find that the root node will always have the highest value in the entire tree. A heap will always be a complete tree, which is where “all levels are completely filled except possibly the last level and the last level has all keys as left as possible” (Geeksforgeeks, 2021, para. 1). Conveniently, since heaps are always complete trees, they can be stored in an array, and the parent and child nodes can be calculated. A heapsort is a “sorting algorithm that takes advantage of a max-heap's properties by repeatedly removing the max and building a sorted array in reverse order” (Lysecky, 2019b, para. 1).

In the heapsort algorithm, we must first turn the list of unsorted numbers into a heap. This is done by looping from the largest internal node back to the root node at index zero. After the array has been sorted into a heap, we now have the largest number at the top of the heap. The heapsort then begins a loop from the last index of the array by switching its value with the first index, placing the highest number at the end of the list. The loop then heapifies all the indexes before it, and repeats the loop on the previous index, until all the numbers are sorted in ascending order.

If we were to pass the list of integers [25, 44, 55, 99, 30, 37, 15, 10, 2, 4] to a heap sort, the heapsort would first turn the given list into a heap.

To heapify a list, we would first identify the last internal node, which is what Rajinikanth defines as a “node which has at least one child” (Rajinikanth, 2020, para. 12). This is calculated by dividing the length of the list by two, and subtracting one, which in this example, gives us the fourth index, with the value of 30. The values of the node’s children are then compared, and if the child’s value is bigger than the parent node, the child and the parent get switched. Since the child node with the value of 4 is less than the parent node with the value of 30, they do not switch. We then heapify the previous node at index three (99). Since the left (10) and right (2) child nodes are both less than 99, no nodes are switched, and we continue to the previous internal node at index two (55). The children of node at index two (55) have the values of 37 and 15, so no indexes are replaced, and we iterate to the previous internal node at index one. The node at index one has the value of 44, and it’s children nodes are at indexes three and four, with the values of 99 and 30, respectively. Since the node at index three (99) is greater than the node at index one, the two nodes switch, and the node at index one is now 99. Lastly, the node at index zero (25) is compared to the nodes at index one and two, and since the node at index one (99) is greater than index zero (25), they switch. The list after being turned into a heap is now [99, 44, 55, 25, 30, 37, 15, 10, 2, 4], and we have the greatest number at index zero. You can view a visual representation of the heap in figure 1 below.

Figure 1.

Binary Tree that represents a heap

Diagram

Description automatically generated

Note. The results represent the list of numbers [25, 44, 55, 99, 30, 37, 15, 10, 2, 4] after being heapified into the list [99, 44, 55, 25, 30, 37, 15, 10, 2, 4] .

After the list is turned into a heap, the heap sort will then perform another loop on the list, starting from the last index, up to the zeroth index. In each iteration, it switches the node that is being iterated with the node at the zeroth index, and heapifying all of the previous indexes of the list. In the first iteration, it is bringing the node with the highest value (99) to the last index, and heapifying all of the previous nodes, giving us the list [55, 44, 37, 25, 30, 4, 15, 10, 2, 99]. The nodes before the last index are now heapified, and the second highest number (55) is now at the zeroth index. and the loop continues with the next iteration. At the end of this loop, we now have the list sorted in ascending order [2, 4, 10, 15, 25, 30, 37, 44, 55, 99].

If the given list was in increasing or in decreasing order, the heap sort would have a time complexity of O(n log n). The reason why is that the first thing that the heap sort does is turn the list into a heap, which would disrupt the increasing or decreasing order of the list, and the heap sort would continue to rebuild the list by placing the highest number to the back of the list, and heapifying the previous nodes, until the list is in ascending order.

**REFERENCES**

GeeksforGeeks. (2021, November 15). Binary heap. <https://www.geeksforgeeks.org/binary-heap/>

Lysecky, R., & Vahid, F. (2019a). 5.12 Heaps. In Design and Analysis of Algorithms. zyBook.

Lysecky, R., & Vahid, F. (2019b). 5.15 Heap Sort. In Design and Analysis of Algorithms. zyBook.

Rajinikanth. (n.d.). Data structures. BTech Smart Class. <https://www.btechsmartclass.com/data_structures/tree-terminology.html>