Priority Queues (Part II)

May 14, 2024

Contents

1 Review

1.1 Priority Queue

So, unlike a ”regular” queue that we learned before, a priority queue is similar in that it has a front and back but the queue is a sorting queue where elements inserted into a priority queue are sorted into the queue. In summary, the fundamental difference between a priority queue and a ”regular” queue is what values are popped from the queue.

There are several ways that can be used to implement a priority queue such as using a dynamic array, sorted dynamic array or an AVL Tree. Something important to note is that the array does not have to be sorted, but having a sorted array will have different time complexities for a priority queue as an unsorted array. A priority queue is a simplification of what we are already used to with these data structures. Depending on which data structure is used, we can emulate a certain sorting algorithm.

2 Heaps

2.1 Heap Property

Heaps are a complete binary tree in principle, but is actually implemented as an array where the parent is always larger than the children. Note that a heap needs to be sorted by this property for it to maintain this heap property.

2.2 Implementation

Consider an implementation with a max heap (the name explains itself):

- **build(x)**: Consider an array X such that $X = 15, 13, 11, 12, 14$. The build will output a sorted array Q such that $Q = 15, 14, 13, 12, 11$. Remember, a priority queue does not have to be a sorted array but a heap needs to be sorted.[1]
- **insert(k)** : When inserting a value to a heap, we add it to the end of the heap. However, when inserting, there is a potential issue that the new value is a large value that ruins the heap property. To counter this we would need to reorder the values in the heap and we would use the function **max_heapify_up(n - 1)**.
- **max_heapify_up(i)**: So consider the same array Q which we will say is a heap, and then add a new largest value to the end of the dynamic array. However, since that value violates the property of a heap, we swap parent and child until we maintain the heap property. It does not have to be a perfected sorted array but simply needs to follow the heap principle in the end where the parent is larger than the children. (Time complexity: $O(log(n))$)[2]
- **delete_max()**: In the case of a max heap, we would swap the last value in the heap with the first value (our maximum value in the heap) and then delete the last element in the heap which is our maximum value and remove it to obtain a constant time removal. Similarly to inserting a value, we would heapify down the heap to maintain the heap property.
- **max_heapify_down(i)**: Very similarly to the up function we defined, there is also a function for down where you compare the parent with the two children and then swap. Since we check both children, the

[1] When you add a bunch of elements unsorted and then max_heapify_up values from the end to the beginning of the array, you will sort the array in $O(n)$ time.

[2] The time complexity is log(n) because a heap is based off of a tree and there are at most log(n) (base 2) swaps needed to restore the property of the heap

swap will not affect the order for the other child of the tree. This function is used to maintain the heap property after deletion of the max value. (Time complexity: $O(log(n))$)

2.3 Heapsort

Consider a priority queue implemented using a heap called Q such that $Q = 16, 14, 15, 12, 11, 13$. We can actually sort the array associated with this by using the function $delete\_max()$. By iteratively deleting the maximum value and storing that maximum value popped, it eventually results in a sorted array. Each $delete\_max()$ is an $O(log(n))$ operation, so since the function is called iteratively (n times), we finally result in a time complexity of $O(nlog(n))$. One important thing to note is that heapsort can be in-

place based off of the implementation of the delete_max. If the delete does not actually remove the value and simply switches values and resizes[3], we eventually result in the same array that is used in the heap that is sorted at the end.

[3]This means you keep in track of the size of the heap with a variable