Priority Queues

May 9, 2024

Contents

1 Review on Direct Access Array sorting

1.1 Counting Sort

Last lecture, we covered the idea of sorting using direct access arrays. The sorting algorithm that creates a direct access array of all possible elements that the array will sort and then using an $O(n)$ iteration through all elements in the array and putting it inside the direct access array created. To get the sorted array, you then iterate through the direct access array looking for slots that have been used (which indicate the element is inside the array) to get the sorted array. This adds an extra complexity of $O(U)$ where U represents the largest possible element in the array and which when combined with the iteration of the array with a complexity of $O(n)$ gives us a combined time complexity of $O(n+U)$.

1.2 Radix Sort

Radix sort is essentially an iterative approach to counting sort. Instead of using keys that represent the values in the range of $[0, 1, \cdots, U]$, we can approach the values as being an ordered set of keys. For example, we can sort a large range of numbers using their digits. Instead of creating a direct access array of all possible values, there can be ”buckets” that contain groups of elements.

Consider an array containing the values: $[0, 10, 100, 999]$. In this example, we'll create a direct access array containing all 10 possible digits for the ones digit. Each slot in the direct access array contains a queue of possible values.

Round 1.
Create a direct access array with 10 digits, each for a possible number for the ones digit. Iterating through the array, we put each element into the direct access array (D), and then grab the array after the the filtering to obtain a new array.

Round 2.
For this round, we'll do the same thing as Round 1 but filtering through the array by the element's tenth digit.

Round 3.
This is the same as previous rounds, except we filter by their hundreds place. After this round the array is sorted. This isn't the most intuitive explanation but, through rounds of filtering we can obtain a sorted array without directly comparing values.

2 Priority Queues

The basic gist of a priority queue is that it is a data structure when elements that go in are automatically sorted into the queue.

2.1 Interface

Functions

2.2 Heaps

A heap is a complete binary tree and a complete binary tree is where every layer is full, except for the last layer where nodes in the last layer are to the left. Heaps are trees that are implemented as arrays.

With the heap property:

- array[i].left = array[2 * i + 1]
- array[i].right = array[2 * i + 2]
- array[i].parent = array[floor($\frac{i-1}{2}$)]

Because of the formulae above, the heap has to be a complete binary tree otherwise these properties will not work. For a heap, there does not need to be a sense of ordering and as long as it represents a complete binary tree, the given heap array formulae will work.

Heaps are trees, implemented as arrays, with the heap property. The heap property is where the parent inside the tree is bigger than or equal to their children and this property enables heap sort.