Lab 12 Binary Heaps Implementation

Learning Outcomes:

After successfully completing this lab the students will be able to:

1. To understand and implement the Binary Heaps
2. To understand the basic insertion and deletion operations on Binary Heaps

Pre-Lab Reading Task:

A binary heap is a complete binary tree in which every node satisfies the heap property which states that:

If B is a child of A, then key(A) ≥ key(B)

This implies that elements at every node will be either greater than or equal to the element at its left and right child. Thus, the root node has the highest key value in the heap. Such a heap is commonly known as a max-heap. Alternatively, elements at every node will be either less than or equal to the element at its left and right child. Thus, the root has the lowest key value. Such a heap is called a min-heap.

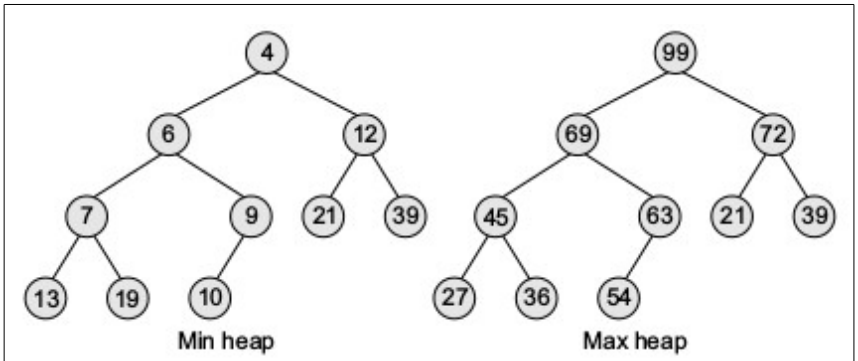

Figure12. 1: Min and Max Heaps

Figure 12.1 shows a binary min heap and a binary max heap. The properties of binary heaps are given as follows:

Since a heap is defined as a complete binary tree, all its elements can be stored sequentially in an array. It follows the same rules as that of a complete binary tree. That is, if an element is at position ***i*** in the array, then its left child is stored at position ***2i*** and its right child at position ***2i+1***. Conversely, an element at position ***i*** has its parent stored at position ***i/2***.

Being a complete binary tree, all the levels of the tree except the last level are completely filled. The height of a binary tree is given as ***log2n***, where n is the number of elements. Heaps (also known as partially ordered trees) are a very popular data structure for implementing priority queues.

A binary heap is a useful data structure in which elements can be added randomly but only the element with the highest value is removed in case of max heap and lowest value in case of min heap. A binary tree is an efficient data structure, but a binary heap is more space efficient and simpler.

Consider a max heap H with ***n*** elements. Inserting a new value into the heap is done in the following two steps:

1. Add the new value at the bottom of H in such a way that H is still a complete binary tree but not necessarily a heap.
2. Let the new value rise to its appropriate place in H so that H now becomes a heap as well.

To do this, compare the new value with its parent to check if they are in the correct order. If they are, then the procedure halts, else the new value and its parent's value are swapped and Step 2 is repeated.

The first step says that insert the element in the heap so that the heap is a complete binary tree. So, insert the new value as the right child of node 27 in the heap.

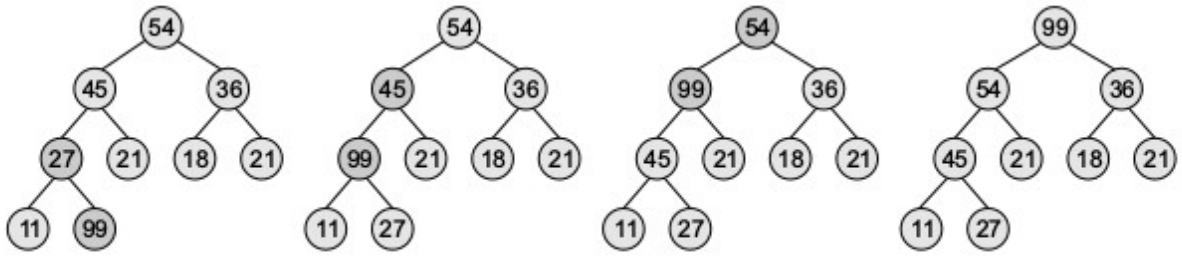

Figure12. 2: Insertion of a node (99) in Max Heap

Now, as per the second step, let the new value rise to its appropriate place in H so that H becomes a heap as well. Compare 99 with its parent node value. If it is less than its parent's value, then the new node is in its appropriate place and H is a heap. If the new value is greater than that of its parent's node, then swap the two values. Repeat the whole process until H becomes a heap.

Deleting an Element from a Binary Heap

Consider a max heap H having n elements. An element is always deleted from the root of the heap. So, deleting an element from the heap is done in the following three steps:

1. Replace the root node's value with the last node's value so that H is still a complete binary tree but not necessarily a heap.
2. Delete the last node.
3. Sink down the new root node's value so that H satisfies the heap property. In this step, interchange the root node's value with its child node's value (whichever is largest among its children). Here, the value of root node = 54 and the value of the last node = 11. So, replace 54 with 11 and delete the last node.

This is illustrated in Figure 12.3 and 12.4 below.

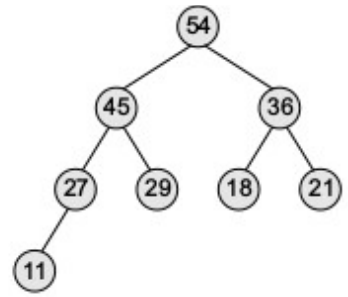

Figure12. 3: The root node always gets deleted

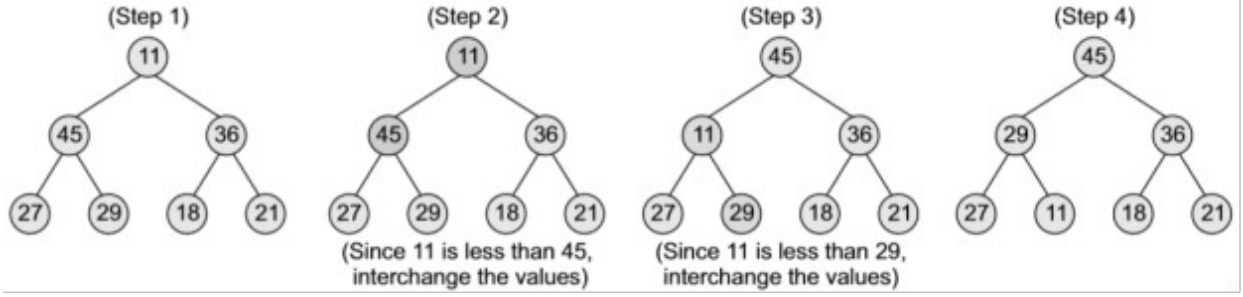

Figure12. 4: Deleting a node from a binary heap

Applications of Binary Heaps

Binary heaps are mainly applied for

1. Sorting an array using heap sort algorithm.
2. Implementing priority queues.

Binary Heap Implementation of Priority Queues

We learned about priority queues. We have also seen how priority queues can be implemented using linked lists. A priority queue is similar to a queue in which an item is dequeued (or removed) from the front. However, unlike a regular queue, in a priority queue the logical order of elements is determined by their priority. While the higher priority elements are added at the front of the queue, elements with lower priority are added at the rear. Conceptually, we can think of a priority queue as a bag of priorities. In this bag you can insert any priority but you can take out one with the highest value. Though we can easily implement priority queues using a linear array, but we should first consider the time required to insert an element in the array and then sort it. We need ***O(n)*** time to insert an element and at least ***O(n log n)*** time to sort the array. Therefore, a better way to implement a priority queue is by using a binary heap which allows both enqueue and dequeue of elements in ***O(log n)*** time.

In-Lab Tasks:

1. Complete the function '`void` **`insert_node`**`(int x, struct heap_struct * H)`' in the skeleton code provided.
2. Complete the function '`void` **`delete_root`**`(struct heap_struct * H)`' in the skeleton code provided.

Post Lab: Study Binomial Heaps and Fibonacci Heaps.