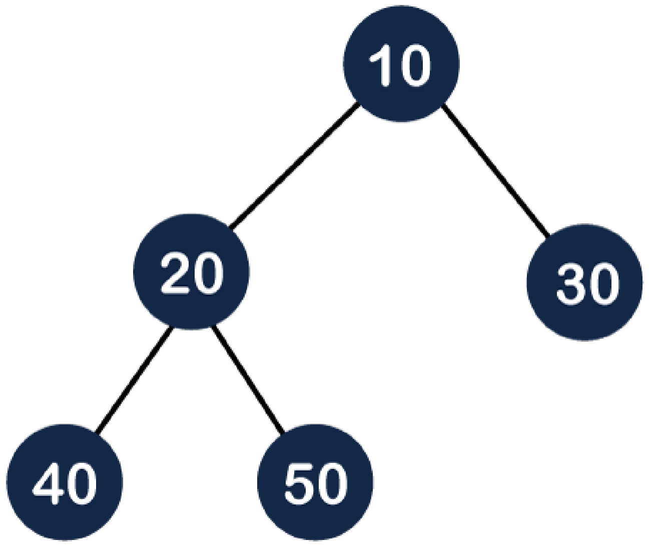
Data Structures and Algorithms

Lecture 32: Heap – Introduction, Insertion

Heap Data Structure

A Heap is a special Tree-based Data Structure in which the tree is a complete binary tree.

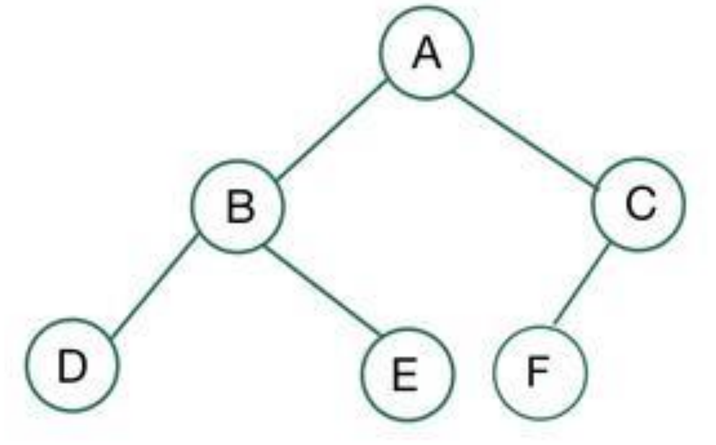


Heap Data Structure

Complete Binary Tree:

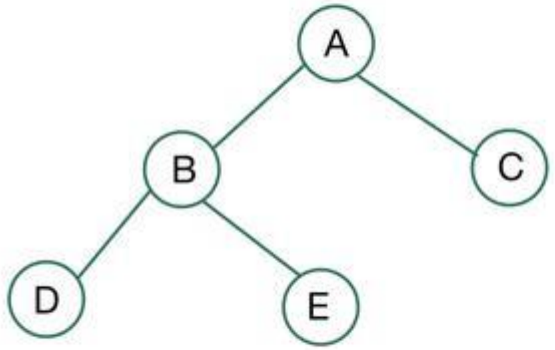
A binary tree is said to be a complete binary tree if all its levels, except possibly the last level, have the maximum number of possible nodes, and all the nodes in the last level appear as far left as possible.

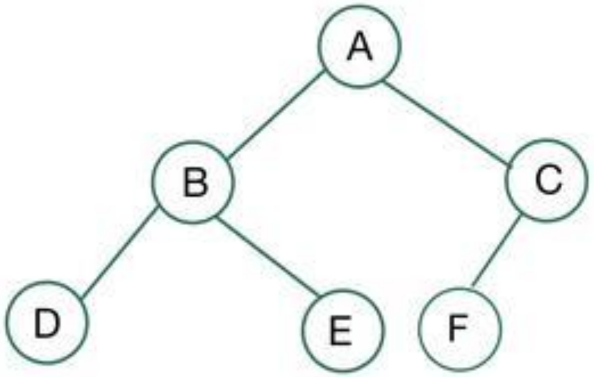
- The leftmost side of the leaf node must always be filled first.
- It isn't necessary for the last leaf node to have a right sibling.

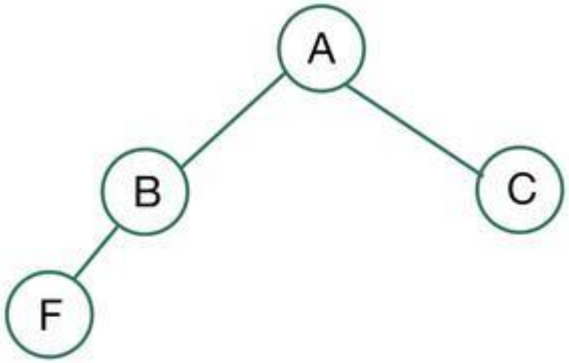


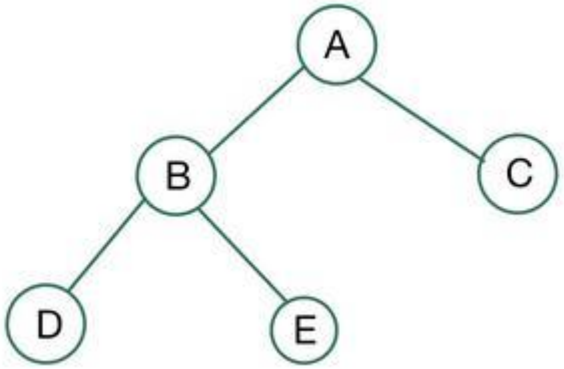
Heap Data Structure

Complete Binary Tree - Examples









Types of Heap Data Structure

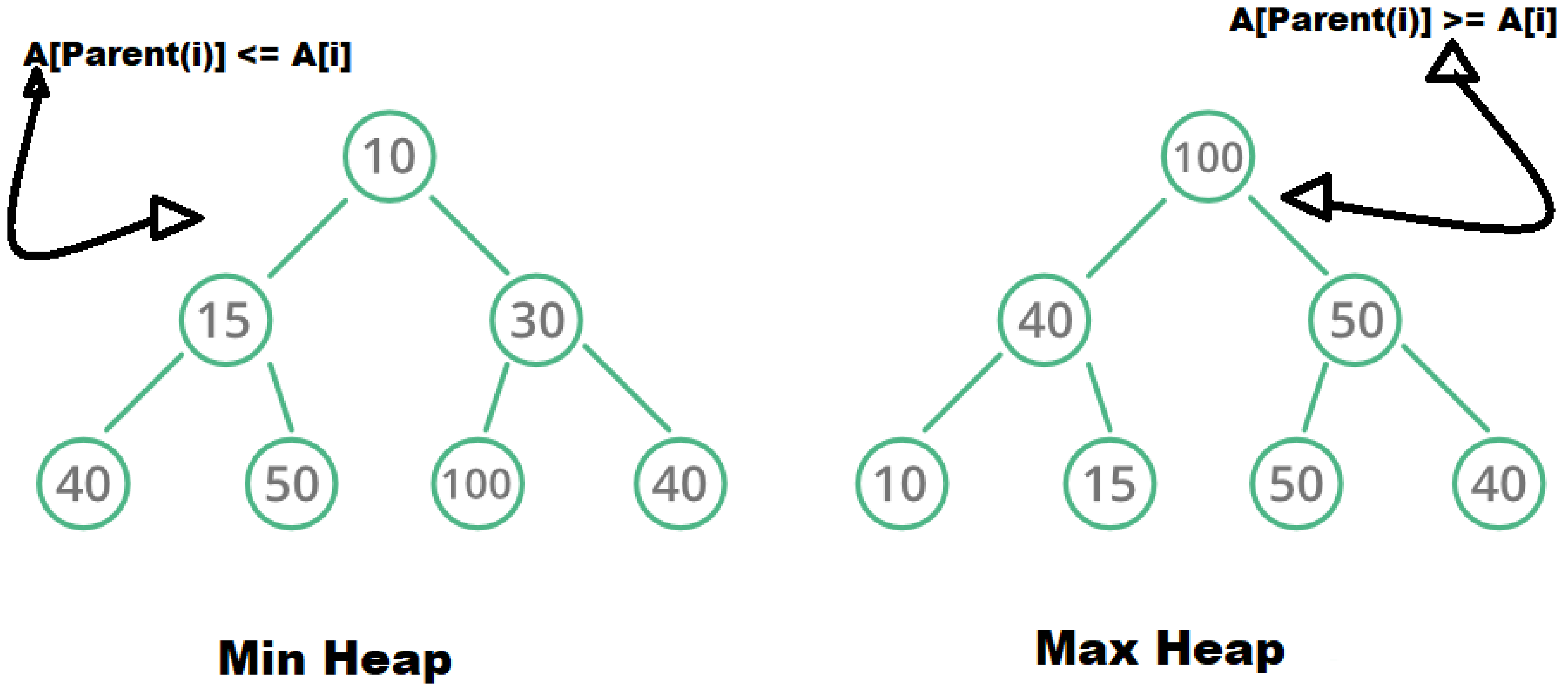
Max Heap –

- The value of the root node must be the greatest among all its child nodes and the same thing must be done for its left and right sub-tree also.
- The total number of comparisons required in the max heap is according to the height of the tree. The height of the complete binary tree is always logn; therefore, the time complexity would also be O(logn).

Min heap –

- The value of the root node must be the smallest among all its child nodes and the same thing must be done for its left and right sub-tree also.
- The total number of comparisons required in the min heap is according to the height of the tree. The height of the complete binary tree is always logn; therefore, the time complexity would also be O(logn).

Heap Data Structure



Properties of Heap

- **Complete Binary Tree:** A heap tree is a complete binary tree, meaning all levels of the tree are fully filled except possibly the last level, which is filled from left to right. This property ensures that the tree is efficiently represented using an array.
- **Heap Property:** This property ensures that the minimum (or maximum) element is always at the root of the tree according to the heap type.
- **Parent-Child Relationship:** The relationship between a parent node at index ‘i’ and its children is given by the formulas: left child at index 2i+1 and right child at index 2i+2 for 0-based indexing of node numbers.

Properties of Heap

- **Efficient Insertion and Removal:** Insertion and removal operations in heap trees are efficient. New elements are inserted at the next available position in the bottom-rightmost level, and the heap property is restored by comparing the element with its parent and swapping if necessary. Removal of the root element involves replacing it with the last element and heapifying down.
- **Efficient Access to Extremal Elements:** The minimum or maximum element is always at the root of the heap, allowing constant-time access.

Heap - Operations

- **Operations supported by min –** heap and max – heap are same. The difference is just that min-heap contains minimum element at root of the tree and max – heap contains maximum element at the root of the tree.

- **Heapify -** It is the process to rearrange the elements to maintain the property of heap data structure. It is done when a certain node creates an imbalance in the heap due to some operations on that node. It takes O(log N) to balance the tree.

For max-heap, it balances in such a way that the maximum element is the root of that binary tree and

For min-heap, it balances in such a way that the minimum element is the root of that binary tree.

Heap – Insertion Operation

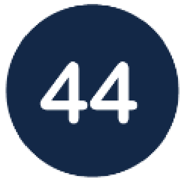
44, 33, 77, 11, 55

Suppose we want to create the max heap tree. To create the max heap tree, we need to consider the following two cases:

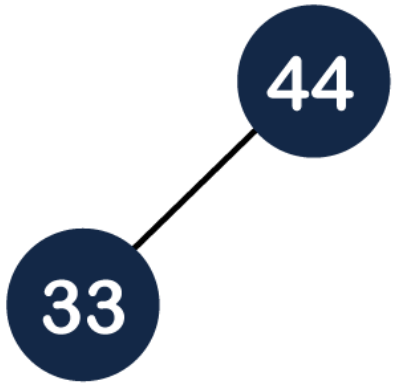
- First, we have to insert the element in such a way that the property of the complete binary tree must be maintained.
- Secondly, the value of the parent node should be greater than the either of its child.

Heap – Insertion Operation

Step 1: First we add the 44 element in the tree as shown below:

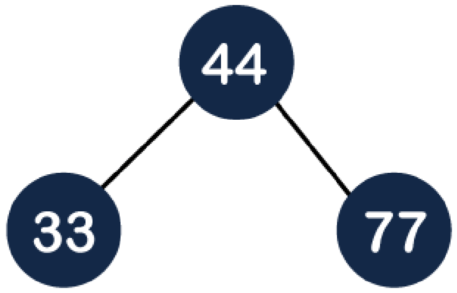


Step 2: The next element is 33. As we know that insertion in the binary tree always starts from the left side so 44 will be added at the left of 33 as shown below:

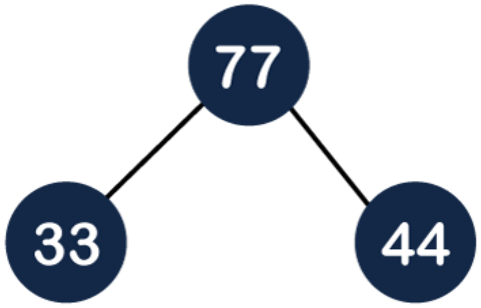


Heap – Insertion Operation

Step 3: The next element is 77 and it will be added to the right of the 44 as shown below:

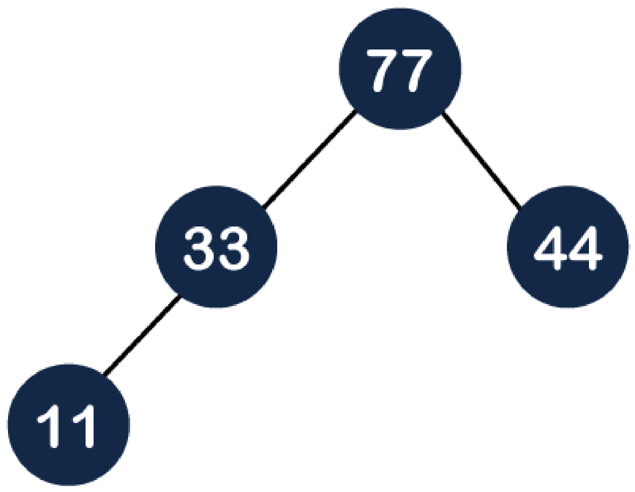


As we can observe in the above tree that it does not satisfy the max heap property, i.e., parent node 44 is less than the child 77. So, we will swap these two values as shown below:



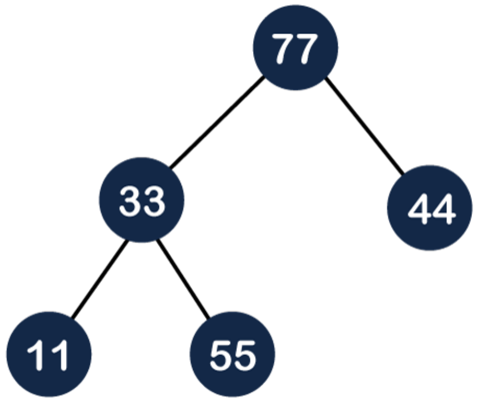
Heap – Insertion Operation

Step 4: The next element is 11. The node 11 is added to the left of 33 as shown below:

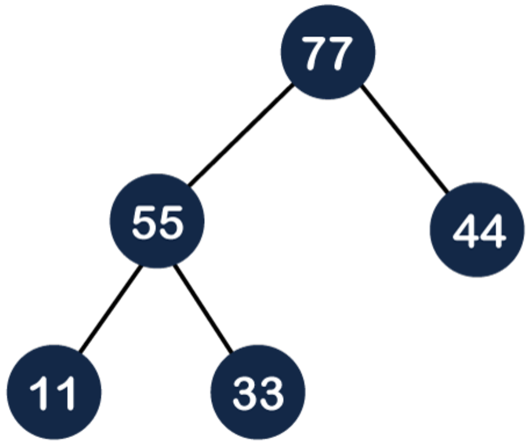


Heap – Insertion Operation

Step 5: The next element is 55. To make it a complete binary tree, we will add the node 55 to the right of 33 as shown below:



As we can observe in the above figure that it does not satisfy the property of the max heap because 33<55, so we will swap these two values as shown below:


77
55
44
11
33