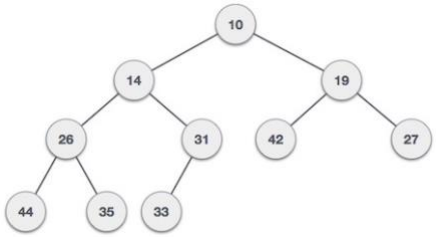
CS 260
Heaps

Heaps

Heaps are binary trees, but they differ from both parse trees and binary search trees. Binary search trees are strongly ordered, the location of a node is specified by its value and when it was added to the tree. All values smaller than a node are to its left and all values greater than or equal to it are to the right. A heap is weakly ordered, that is any path from the root to a leaf is descending (for a max heap) or ascending (for a min heap). Heaps are not used for organizing data for searching, only for keeping track of the largest or smallest value.

This is an example of a heap with the smallest value (10) at the root and all paths ascending from root to leaf.



A heap follows the following rules:

- it is a binary tree
- the largest (or smallest for a min-heap) value is at the root
- it is complete (each layer in the tree is full, except the bottom-most layer which is filled from left to right)
- it is weakly ordered (every path from the root to a leaf is descending — or ascending for a min heap)

Adding a new value to heap follows the following algorithm (for max heap, root largest value):

1. add the new value at the next location in the array (next empty location in bottom row)
2. starting with newly added value
    - compare it with its parent
    - if parent is smaller, swap
    - repeat until parent is no longer smaller or root is reached

Removing the largest value follows the following algorithm:

1. remove and save the root value
2. move the last value (rightmost value in bottom row) to root
3. starting with new root
    - compare with its children
    - if either child is larger, swap with largest child
    - repeat until no longer smaller than children or reach a leaf

Typically, the algorithm for restoring the heap after adding an item is referred to as bubble up, since the new value bubbles up along the path to the root until it reaches its proper place. Similarly, the algorithm for restoring after removing a value is trickle down since the value starts at root and trickles down the paths until it reaches its final resting place.

Since the height of the tree is log2 of the number of nodes and each of these algorithms has at a maximum number of steps equal to the height, it is easy to see that they both are O(log N) algorithms.

Implementation on an array

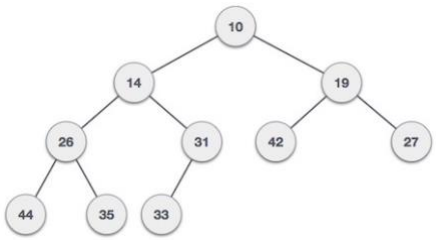
The fact that heaps are complete means that implementing them using arrays is effective. The root is placed at index one with its two children at 2 and 3. Then the location of a child can be calculated as left child is 2 * index and right child is 2 * index + 1. The parent is at index / 2.

This allows for a simple implementation where only the values need to be sorted in the array.

Implementation with references for parent and children

Since a heap is a binary tree, it can be implemented in the same manner as a parse tree or a BST, that is by using a node that contains references to parent and left and right children. The only issue is how to find where to place the next value. This example is provided for information only, normally heaps are implemented on an array since it is more efficient, and the code is much simpler. This is particularly true when implementing heapsort.

Consider the heap image from earlier:



It is clear from looking at the image that the next node to be added will be a left child of the node containing 31. To determine how to find this node from the root, we need to use a numbering of the nodes. Start with the root at 1, then its left child is 2 and its right child is 3. This is simply a count of the number of nodes. If we have 10 nodes in the tree, then the next node is added at location 11. The binary representation of 11 is 1011. Dropping the leading bit gives a pattern of 011, this can be translated into go left, then go right, then go right as the program moves down the tree. This provides the location to add the next value.

Priority Queues

Since a heap allows removing the smallest (or largest) value or adding a new value in O(Log N) time, it is the optimal data structure to use for implementing a priority queue. The solution is to use class composition and create an instance of a heap inside the priority queue class and then use its methods to provide the peek, pop, and push methods of the priority queue.