Heaps and Priority Queues

Joyen Benitto

Heaps and its ADT?

A heap is an ADT which encapsulates the following set of operations.

- insert ($insert(S, x)$): Insert a element x into a set S.
- max ($max(S)$): Returns element of S with the largest key.
- max and pop ($extract\_max(S)$): Extracts the maximum and pops the max value.
- Increase key ($increase\_key(S, x, k)$): Increase the value of x's key to the new value k.

A heap is primarily an array data structure but it is visualized or treated as a nearly complete balanced binary tree.

A balanced binary tree is $\equiv a$ binary tree where the height difference between the left and right subtrees for every node is at most 1 $\equiv$, ensuring the tree stays short and wide, not tall and skinny, which keeps operations (search, insert, delete) efficient, typically O(log n) time, unlike unbalanced trees that can degrade to O(n).

So conceptually a heap is a balanced tree where each node can have up to two children, a left and a right child. The keys in such a binary tree are said to be in *heap order*.

Heap order: For every element v , at a node i , the element w at i's parent satisfies $key(w) \leq key(v)$

So let us talk a little bit about the implementation either we can have every node to have the elements value, the key and the pointer to the next two children node or if the size of the elements is know we can just hold all the elements in the array and just treat the indices i as the elements of the heap and the children $leftchild(i)$ and $rightchild(i)$ are $2i$ and $2i + 1$ respectively and of-course the $parent(i)$ is at $i/2$.

When $n < N$ we will use the first n elements of the array we use $length(H)$ to denote the length of the heap.

Heaps are about decision and not order

Implementing the Heap Operations

The heap element with the smallest key is always on top and is at the root