Lecture 2, Sep 13, 2023

Definition

An *abstract data type* (ADT) describes an object and which operations you can apply to it.

A *data structure* is a particular implementation of an ADT.

Max-Heaps

Definition

Given a set of S elements with keys (i.e. priority) that can be compared, a *priority queue* has the operations:

- $\textsc{Insert}(S, x)$: insert element x in S
- $\textsc{Max}(S)$: returns an element of highest priority in S (note there may be multiple elements with the same priority)
- $\textsc{ExtractMax}(S)$: returns the max element and removes it from S

- Priority queues can be implemented in a variety of ways, e.g. with linked lists or sorted arrays, but max-heaps are a particularly efficient implementation that offers $\Theta(\log n)$ operations

Definition

In a (binary) *max-heap* of n elements, the elements are stored in a complete binary tree such that the max-heap property holds, i.e. the priority of each element is greater than or equal to all its children.

- Note: in a complete binary tree, all levels are filled except the last one, and the last level is filled starting from the left
  - The height of a complete binary tree with n nodes is $\lfloor \log_2 n \rfloor$
- For a given sequence of elements, there is no unique max-heap configuration
- Max-heaps (or in general any complete binary tree) can be stored very efficiently in a simple array by laying out its elements from top to bottom and left to right
  - If an element is at index i, then its children will be at indices $2i$ and $2i + 1$ respectively (note we're using 1-based indexing; with zero-based indexing, the children will be at $2i + 1$ and $2i + 2$)
    - To go back to the parent, we take $\left\lfloor \dfrac{i}{2} \right\rfloor$
  - The size of the heap is tracked so we know where the array ends
- To insert an element, we add the element at the end, increase the heap size, and restore the heap property
  - To restore the heap property, compare the current element with its parent, and if it has higher priority, then swap the two elements; repeat all the way until the element no longer has greater priority than its parent, or it has reached the root
  - Since we have to do this for at most all levels of the tree, this operation is $\Theta(\log n)$ (since the tree's height is $\lfloor \log n \rfloor$)
- To get the max element, we simply return the root of the tree, which is a constant time operation
- To extract the max element, we extract the root of the tree and replace the now empty root with the last element, and restore the heap property
  - To restore the heap property, compare the current element with its children, and if it is smaller than any of its children, swap it with the bigger child; repeat until the element is bigger than its children, or becomes a leaf
  - This again goes through at most all levels of the tree, making it $\Theta(\log n)$ complexity
- Some example applications:

- HeapSort: take an array, make it a heap, and extract the max n times until the array is empty; this makes for a simple $\Theta(n \log n)$ sorting algorithm