CSC263 - Week 2, Lecture 2

Cristyn Howard

Friday, January 19, 2018

- All heaps must satisfy the heap order property.
  - in a MAX heap: values of children of node X are *less than* or equal to the value of node X
  - in a MIN heap: values of children of node X are *greater than* or equal to the value of node X
- If there is one node (in a max heap) that violates the heap order property (i.e. it's children are larger than it), recursively compare the value of node X to the value of it's children and swap X with it's largest child until the heap order property is restored.
- Max-Heapify(A,i): subprocedure used in INSERT and EXTRACT
  - *Inputs:* array A, index in A
  - *Pre-condition:* subtrees rooted at the left & right child of the node at i are heaps
  - *Post-condition:* the subtree rooted at index i is a heap.
  - ```
    MaxHeapify(A,i):
    L = LEFT(i), R = RIGHT(i);
    if L ≤ heapsize(A) and A[l] > A[i]
        largest = L
    else
        largest = i
    if R ≤ heapsize(A) and A[r] > A[largest]
        largest = R
    if largest ≠ i
        swap(A[i],A[largest])
        MaxHeapify(A,largest)
    ```
  - Max-Heapify $\in O(\log n)$
- Build-Max-Heap:
  - First, store elements in an array representing a complete binary tree.
  - Then, iterate from leaf to root performing MaxHeapify. Note that it is not necessary to perform MaxHeapify on leaf nodes, as they by definition preserve the heap order property and have no children to swap with. Because the last half of the nodes in an array are leaf nodes*, MaxHeapify is called starting from $\left\lfloor \frac{|A|}{2} \right\rfloor$.
  - Build-Max-Heap(A): for $i = \left\lfloor \frac{|A|}{2} \right\rfloor$ to 1, MaxHeapify(A,i).
  - Intuitively, because Build-Max-Heap calls an $O(\log(n))$ function n times, it would be in $O(n \log(n))$, but it does not actually iterate through the whole height of the tree n times!
    ```

- Proof that Build-Max-Heap is $\in O(n)$:
  - each node at depth d has a height of at most $h - d$ $\therefore$ the heapification cost of each node is at most $h - d$
  - $\therefore$ max heapification cost of each level $= 2^d(h - d)$
  - $\therefore$ max heapification cost of the whole tree $= \sum_{d=0}^{h-1} 2^d(h - d)$
  - set $i = h - d$, write $\sum_{i=1}^{h} 2^{h-i} i = 2^h \sum_{i=1}^{h} \frac{i}{2^i} \leq (2^h) \sum_{i=1}^{\inf} \frac{i}{2^i} = (n)2$

- Deleting from Binary Search Trees
  - Case A: X has no children
    - set parent's pointer to null, removed
  - Case B: X has one child
    - set parent's pointer to X's child, removed
  - Case C: X has two children
    - Successor(X): smallest node bigger than X, leftmost node in the right subtree of X
    - set x to value of it's successor, delete successor (reduces to case A or B)