Complexity of Heapifying

Andrzej

January 18, 2006

Assume a collection $Q$ of $n$ elements comparable by a total ordering relation to be organized into a priority queue implemented as a binary heap. Wlog., the elements of $Q$ reside as entries indexed $1..n$ of an array $A$. There are two approaches to “heapify” $A$: one is top-down (by iterated insertions), another: bottom-up (you can think of it as somewhat related to dynamic programming). The former is performed by a series of “sift-up($i$)” operations, repeated for $i = 1..n$. The latter is implemented by the series of “join“($i$)” operations (described in class) repeated for $i = n..1$ (the operations dealing with elements without descendants in the tree are “no-ops”).

What are complexities of those approaches?

Since the complexity of sift($i$) $\in \Theta(\log i)$ (the depth of element $i$), complexity of top-down

$$\text{top-down}(n) = \sum_{1\leq i\leq n} \text{sift}(i) \geq \sum_{n/2\leq i\leq n} \log(i) \geq n/2 \log(n/2) \in \Omega(n \log(n))$$

(notice the common trick of limiting the considerations only to “large” values of $i$.)

The complexity of join($i$) is $\mathcal{O}(\log(n) - \log(i))$ (the height of $i$). Thus the complexity of repeated joins is

$$\text{bottom-up}(n) = \sum_{1\leq i\leq n} \text{join}(i) = \sum_{1\leq k\leq \log(n)} k \cdot 2^{\log(n) - k} = \sum_{1\leq k\leq \log(n)} kn/2^k = n \sum_{1\geq k\geq \log(n)} k/2^k$$

Somewhat surprisingly, the last sum is constant, as it can be represented as

$$\sum_{1\leq k\leq \log(n)} \left( \frac{1}{2^k} \sum_{1\leq m} \left( \frac{1}{2^m} \right) \right)$$

The inner sum is less than 1 (even without bounding $m$ from above) and therefore the expression is a constant times $n$, ie., linear!